

A Multimedia Device and Stream Management Architecture Based on Distributed-Object Computing Technology

Eine Architektur für das Management von multimedialen Geräten und Strömen unter Einsatz
verteilter Objekttechnologie

Dem Fachbereich 18 der
Technischen Universität Darmstadt
zur Erlangung der Würde eines
Doktor-Ingenieurs (Dr.-Ing.)
vorgelegte und genehmigte Dissertation

von

Dipl.-Ing. Reinhold Eberhardt

geboren am 16. Januar 1959 in Ulm.

Referent:	Prof. Dr.-Ing. Ralf Steinmetz
Korreferent:	Prof. Dr. techn. Joachim Swoboda
Tag der Einreichung:	8. November 2000
Tag der mündlichen Prüfung:	6. Februar 2001

D 17
Darmstädter Dissertationen

Abstract

Recent years have seen a veritable boom in multimedia applications. Due to the ubiquitous computing paradigm and evolution, there is now a great need for distributed multimedia applications, in particular. However, the majority of these multimedia applications place high demand on quality of service (QoS) with respect to the communication and end systems. Until recently, most of these applications have been stand-alone applications: generally proprietary solutions which are either tailor-made for a specific communications systems or often extremely difficult to extend to other multimedia devices than those they are intended for. Hence, a prime motivation for this work lies in the recent giant steps taken in development of multimedia and communication systems and also in the advances made in designing and realizing distributed systems in general.

It would be most effective if programmers were able to tackle the development of distributed multimedia applications using exactly the same, tried and tested methods as when developing conventional distributed applications, without having to worry about the data transfer between the various multimedia devices and the communications system or between multimedia devices. So, in this thesis, I argue that a distributed multimedia application can be designed by using the same principles as for any other distributed applications without sacrificing the need for quality of service. And, the aim of this thesis is to design and realize a multimedia device and stream management architecture which is based on distributed-object computing technology and, at the same time, ensures the quality of service required.

This work was done in a field of application where the major concern was to support an organization's workforce in its life-long professional learning process through a mixture of traditional classroom-based face-to-face training, teletraining, and self-paced training programs such as CD ROM- or Web-based training. The requirements set for such applications are presented in this dissertation, because the applications themselves belong to a superior example of a distributed multimedia application.

The first part of this work defines the notions of multimedia, communication systems, quality of service, and object-based distributed systems as the foundation of this work. Then, I go on to present an analysis of related work in research and standardization venues and identify a number of limitations in the work recorded in literature.

The management architecture described in this dissertation is characterized by its openness, the fact that it may be efficiently integrated into heterogeneous systems, its scalability and, in particular, by its versatility. In order that the architecture might enjoy wide-spread deployment, we designed generic operating system-independent programming interfaces for the multimedia devices to be

utilized and the quality-of-service-aware communications systems currently in use and implemented them in a prototype.

A major contribution of this work is the design, implementation, and evaluation of the device and stream management layer of the architecture. Various multimedia devices and quality-of-service-aware communications systems were integrated into the architecture, and they demonstrate how simple it is when using the framework to rapidly and flexibly integrate new devices, which are then very easy to manage. All the different parts of the management architecture were designed, specified, and documented using UML as the formal modeling language. With the aid of an integrated tool, we were able to bring the techniques and components developed together in a prototypical implementation of the system, thus achieving concrete validation of the approach taken in this thesis.

The research work discussed in this dissertation shows that applying the concepts of distributed application design to distributed multimedia applications by using distributed-object technology and by separating the transport stream from the control and management of the stream is a feasible option. It was proven both in the lab and in the production environment that the basic approach underlying this work is indeed viable and, more importantly, that even high-volume video streams could be handled without difficulty.

Zusammenfassung

In den letzten Jahren wurden eine Vielzahl von multimedialen Anwendungen auf den Markt gebracht. Durch die Hinwendung zur und durch das Paradigma der Ubiquität von Rechnern, besteht die Notwendigkeit für verteilte, multimediale Anwendungen. Die Mehrzahl dieser multimedialen Anwendungen stellen Anforderungen hinsichtlich der Dienstgüte von Kommunikations- und Endsystemen. Bisher handelte es sich hierbei meistens um alleinstehende Anwendungen, die nur bedingt miteinander kommunizieren konnten. Für verteilte, multimediale Anwendungen werden überwiegend proprietäre Lösungen entwickelt, die entweder eng mit einem spezifischen Kommunikationssystem gekoppelt sind oder nur schwer um andere multimediale Endgeräte erweiterbar sind. Einerseits wird diese Arbeit durch die vor kurzem erfolgten Fortschritte bei Multimedia- und Kommunikationssystemen und andererseits durch die Fortschritte beim Entwurf und der Implementierung von verteilten Systemen motiviert.

Analog zur Entwicklung von verteilten Anwendungen sollte ein Anwendungsprogrammierer mit den gleichen vertrauten Methoden verteilte, multimediale Anwendung entwickeln können, ohne daß er sich um die Kommunikation der multimedialen Daten zwischen Endgerät, Kommunikationssystem und Endgerät kümmern muß. Deshalb vertrete ich in dieser Arbeit den Standpunkt, daß verteilte, multimediale Anwendungen mit den gleichen Prinzipien wie andere verteilte Anwendungen entworfen und implementiert werden können, ohne die Notwendigkeit für Dienstgüte zu verlieren. Das Ziel dieser Arbeit ist der Entwurf und die Implementierung einer Architektur für das Management von multimedialen Geräten und Kommunikationströmen unter Einsatz verteilter Objekttechnologie und unter Gewährleistung der notwendigen Dienstgüte.

Der Anwendungshintergrund aus dem diese Arbeit herrührt, ist den lebenslangen Lernprozeß der Mitarbeiter eines Unternehmens durch ein didaktisches Konzept, welches aus einer adäquaten Kombination von traditionellem Präsenzunterricht, Telepräsenzs Schulung und selbst gesteuertem Lernen wie CD- oder Web-basiertem Training besteht, zu unterstützen. Die Anforderungen derartiger Anwendungen werden in dieser Arbeit dargestellt, weil sie hervorragende Beispiele für verteilte multimediale Anwendungen sind.

Im ersten Teil dieser Arbeit werden die grundlegenden Begriffe in den Bereichen Multimedia, Kommunikationssysteme, Dienstgüte und objektorientierte, verteilte Systeme definiert. Weiterhin werden verwandte Arbeiten aus der Forschung und Standardisierung analysiert und eine Reihe von Beschränkungen in den existierenden Arbeiten identifiziert.

Die in dieser Arbeit beschriebene neuartige Managementarchitektur zeichnet sich durch ihre Offenheit, Integrationsfähigkeit für heterogene Systeme, Skalierbarkeit und vielseitige Einsetzbarkeit aus. Damit die Architektur möglichst universell einsetzbar ist, wurden generische, betriebssystemunabhängige Programmierschnittstellen für Multimediaendgeräte und dienstgütefähige Kommunikationssysteme konzipiert und prototypisch implementiert.

Ein wesentlicher Beitrag dieser Arbeit ist der Entwurf, die Implementierung und die Evaluierung der Geräte- und Strommanagementschicht der Architektur. Verschiedene Multimediaendgeräte und dienstgütefähige Kommunikationssysteme wurden eingebunden und erbringen den Nachweis, daß es durch die Verwendung des Rahmenwerks möglich ist, schnell und flexibel neue Geräte, die dann auf einfache Art und Weise steuerbar sind, zu integrieren. Alle Teile der Managementarchitektur wurden unter Verwendung der formalen Modellierungssprache UML entworfen, spezifiziert und dokumentiert. Mit Hilfe eines integrierten Werkzeugs wurden die entwickelten Verfahren und Komponenten prototypisch implementiert, um den Ansatz dieser Arbeit zu validieren.

Die Arbeit zeigt, daß es möglich ist, die Konzepte für den Entwurf von verteilten Anwendungen unter Verwendung verteilter Objekttechnologie durch die Trennung des Transportstroms von der Steuerung und dem Management dieses Stroms auf verteilte, multimediale Applikationen anzuwenden. Unter Laborbedingungen und im Produktionsumfeld konnte nachgewiesen werden, daß der grundlegende Ansatz dieser Arbeit tragfähig ist und selbst hochvolumige Videoströme problemlos gehandhabt werden können.

Acknowledgements

First, I would like to thank my former head of department at DaimlerChrysler AG, Bernhard Hohlfeld, and my supervisor at the Technical University of Darmstadt, Ralf Steinmetz, both of whom encouraged me to start this thesis, for their continuing help and support in concluding this work. Secondly, thanks go to all of the research staff and students in the Department of Communication Technology at DaimlerChrysler and the University of Darmstadt for their fruitful discussions and support of this work. I would especially like to acknowledge Christian Rueß for his readiness to participate in innumerable intensive discussions with me. Additional special thanks are due to Jochen Metzler, Christian Wilk, Rolf Sigle, Michael Wolf, Rainer Rusnak, Matthias Rueß, Mathias Wiegard, and Matthias Brusdeylins at DaimlerChrysler, Michael Lippert, and Carsten Griwodz at the University of Darmstadt, Lars Wolf at the University of Karlsruhe, and Stephan Fischer at GMD IPSI Darmstadt.

I am particularly grateful to my second supervisor Joachim Swoboda for his immediate willingness to support this work. This thesis would not have been finished so rapidly without the backing of Ralf-Guido Herrtwich and Joachim Beer, who gave me the opportunity to finalize my work at the International Computer Science Institute (ICSI) at the University of California at Berkeley.

Also many thanks are due to Larissa Glaser for reviewing and improving my English.

Finally, I owe my deepest gratitude to my family – my wife Angelika and my children Carolin, Selina and Leonie – for their continued support and encouragement – and their amazing patience. Thank you very, very much.

Table of Contents

1	INTRODUCTION.....	15
1.1	Problem Statement.....	15
1.2	Research Goal.....	16
1.3	Outline of the Dissertation.....	17
2	TERMINOLOGY, PRINCIPLES, AND BASIC CONCEPTS.....	19
2.1	Multimedia and Communication Systems.....	19
2.2	Quality of Service	19
2.2.1	The Dimensions of the QoS Problems.....	21
2.2.2	The Best-Effort-to-Static-to-Dynamic Spectrum	22
2.2.3	QoS Requirements, Agreements, and Contracts	22
2.2.4	Mapping of QoS Requirements.....	22
2.2.5	Resource Management, Scaling, Filtering, and Application Adaptation.....	23
2.2.6	Pricing.....	23
2.3	Object-based Distributed Systems.....	23
2.3.1	OMG OMA	23
2.3.2	Java RMI	24
2.3.3	DCOM.....	25
2.3.4	Comparison of Distributed-Object Technology.....	26
2.4	Application Programming Interfaces.....	27
2.5	Video Compression	27
2.6	Summary	28
3	STATE OF THE ART AND RELATED WORK.....	29
3.1	QoS in Standards.....	29
3.1.1	ISO Open Systems Interconnection.....	29
3.1.2	ITU-T and ATM Forum	31
3.1.3	UNIX ATM API.....	33
3.1.4	IETF.....	35
3.1.4.1	Internet Traffic Management.....	35
3.1.4.2	Integrated Internet Services Architecture (IntServ)	36
3.1.4.3	Internet Differentiated Services Architecture (DiffServ).....	39
3.1.4.4	QoS Policies.....	43
3.1.4.5	IETF End-to-End QoS Model.....	45
3.1.5	Microsoft Windows Socket API Version 2.....	45
3.1.6	RSVP-aware Communication APIs.....	47
3.2	QoS Architectures.....	47
3.2.1	QoS-A.....	48
3.2.2	OMEGA.....	49
3.2.3	Heidelberg QoS Model.....	50
3.3	Related Work (Stream Control and Management).....	51
3.3.1	ORB-based Approaches	52
3.3.1.1	OMG A/V RFP.....	52
3.3.1.2	TAO	54
3.3.1.3	NEC C&C Research Laboratories.....	54
3.3.1.4	Extended Integrated Reference Model (XRM).....	54
3.3.1.5	TINA.....	56
3.3.1.6	Open Signaling and Architecture Interest Group	56

3.3.2	IETF.....	56
3.3.3	Development Environments for Multimedia Applications	57
3.3.3.1	Java Media Framework (JMF).....	57
3.3.3.2	Configurable INtEgrated Multimedia Architecture (CINEMA).....	58
3.3.3.3	Distributed Multimedia Object Services (DMOS)	58
3.3.4	DSM-CC and DAVIC	59
3.4	Active Networks and QoS.....	59
3.5	Summary	60
4	REQUIREMENTS ANALYSIS.....	63
4.1	Learning Process	63
4.1.1	Learning Management.....	64
4.1.2	Analysis, Design, and Production of Content.....	65
4.1.3	Learning.....	66
4.1.4	Assessment and Feedback.....	68
4.1.5	Roles in the Training Development Process.....	68
4.2	Content Management and Delivery	69
4.3	Summary	72
5	MANAGEMENT ARCHITECTURE	75
5.1	System Architecture	75
5.2	Multimedia Device and Stream Management Architecture.....	76
5.3	Architectural Integration of Real-Time Transport Systems	78
5.4	Integration of Multimedia Devices.....	79
5.5	System Interaction	80
5.6	Open Framework.....	82
5.7	Application Areas and Summary.....	83
6	MULTIMEDIA DEVICE AND STREAM CONTROL	85
6.1	Architecture.....	85
6.2	Communication Model	87
6.3	Object Model.....	90
6.4	Buffer Management.....	92
6.5	JMF Integration.....	92
6.6	Summary	93
7	MANAGEMENT SERVICE.....	95
7.1	Architectural Concept of the Management Layer	95
7.2	Interaction with the Device Layer.....	96
7.3	Graphical User Interface	97
7.4	Summary	100
8	IMPLEMENTATION.....	101
8.1	Software Structure	101
8.2	CORBA Interfaces.....	102
8.3	Java Implementations.....	102
8.4	Device Layer	103
8.4.1	JNI Interfaces	103

8.4.2	Interoperability of Java and Native Interfaces.....	105
8.4.3	Buffer Management.....	105
8.4.4	Device Binding.....	106
8.4.5	Push and Pull Threads	107
8.5	Programming Tools.....	107
8.6	Summary	107
9	EXPERIMENTS.....	109
9.1	Application Area	109
9.2	Experimental Environment.....	109
9.2.1	Hardware.....	109
9.2.2	Software	110
9.3	Summary	111
10	CONCLUSIONS AND FUTURE WORK.....	113
10.1	Conclusions.....	113
10.2	Thesis Contribution	113
10.3	Future Work.....	114
11	APPENDIX	117
11.1	Video Compression	117
11.1.1	H.261.....	117
11.1.2	MPEG-1.....	117
11.1.3	MPEG-2.....	118
11.2	ATM Classes of Service	121
11.3	Internet RFCs	122
11.4	Multicasting.....	123
11.5	Microsoft Windows Socket API Version 2 Flowspec	125
11.6	Object Model in Detail.....	128
11.7	IDL Interface Definition	132
11.8	Lab Configuration.....	136
11.9	Abbreviations and Acronyms.....	138

List of Figures

Figure 2.1: OMA Reference Model – Interface Categories.....	24
Figure 3.1: Reference Model for Native ATM [ATMF96d]	32
Figure 3.2: UNIX ATM Architecture Using XTI, XSocket, and DLPI ATM APIs [ATMF97e] ...	34
Figure 3.3: IntServ Protocols and Mechanisms	37
Figure 3.4: RSVP	38
Figure 3.5: DiffServ Traffic Classification and Conditioning	41
Figure 3.6: DiffServ Codepoints (DSCP).....	41
Figure 3.7: IETF End-to-End QoS Model.....	45
Figure 3.8: Winsock 2 as the Central Element of WOSA.....	46
Figure 3.9: Protocol-independent Winsock 2 QoS (“Flowspec”).....	46
Figure 3.10: QoS-A.....	49
Figure 3.11: OMEGA QoS Broker	50
Figure 3.12: Heidelberg QoS Model	51
Figure 3.13: Example RFP Stream Architecture	53
Figure 3.14: Example Bi-Directional Video Connection as per the OMG A/V Streaming Specification	53
Figure 3.15: Extended Integrated Reference Model (XRM)	56
Figure 3.16: JMF Architecture.....	57
Figure 3.17: DMO Server Architecture.....	59
Figure 4.1: Process Model.....	64
Figure 4.2: Teletraining and Distance Learning Scenario.....	70
Figure 5.1: System Architecture.....	76
Figure 5.2: Multimedia Device and Stream Management Architecture.....	77
Figure 5.3: Java QoS API.....	79
Figure 5.4: Multimedia Device API	80
Figure 5.5: Example Configuration of MMDevices and FEPs (Step 1).....	81
Figure 5.6: Example Configuration of MMDevices and FEPs (Part 2)	82
Figure 5.7: Example Configuration of MMDevices and FEPs (Part 3)	82
Figure 6.1: Multimedia Device and Stream Control Architecture.....	86
Figure 6.2: Simplified Object Model	86
Figure 6.3: Communication Between Base Devices.....	87
Figure 6.4: Example with One Source and Two Sink Objects	88
Figure 6.5: Object Model of the Streaming Architecture.....	91
Figure 6.6: Integration of JMF Player Functionality.....	93

Figure 7.1: Main Elements of the Architecture at the CORBA Level	95
Figure 7.2: Multiple Connection with Different Devices and Streams	96
Figure 7.3: Basic Management GUI.....	98
Figure 7.4: Example List of Available Factory Objects	98
Figure 7.5: New Connection Dialog (1).....	99
Figure 7.6: New Connection Dialog (2).....	99
Figure 7.7: New Connection Dialog (3).....	99
Figure 7.8: Connection Between Two End Systems	99
Figure 8.1: Use of Java/CORBA and Native Code	101
Figure 8.2: Java QoS API and Connection with Multimedia Devices	102
Figure 8.3: Integration of Devices with JNI.....	104
Figure 8.4: Base Device Binding	105
Figure 8.5: Interoperation of Java and Native Interfaces	105
Figure 8.6: Internal and External Binding.....	106
Figure 8.7: Base Device with Push and Pull Threads.....	107
Figure 9.1: Experimental System.....	110
Figure 9.2: Experimental Configuration.....	111
Figure 11.1: MPEG-2 Data Streams.....	120
Figure 11.2: Unicast Transmission.....	124
Figure 11.3: Multicast Transmission	124
Figure 11.4: Broadcast Transmission	125
Figure 11.5 Object Model.....	129
Figure 11.6: Object Model – Base Device, Flow Endpoint and Multimedia Device.....	130
Figure 11.7: Object Model – Flow Endpoints.....	131
Figure 11.8: Object Model – Multimedia Devices	132

List of Tables

Table 2.1: Functional Comparison of CORBA and RMI.....	26
Table 3.1: OSI Performance-oriented QoS Parameters.....	30
Table 3.2: OSI Non-Performance-oriented QoS Parameters	31
Table 3.3: JMF 2.0 Media Formats [Sun99]	58
Table 6.1: Case Study with One Source and Two Sink Base Devices.....	89
Table 11.1: H.261 and MPEG-1 Parameters	118
Table 11.2: MPEG-2 Profiles	119
Table 11.3: MPEG-2 Levels	119
Table 11.4: Examples of Frame Formats and Bandwidths Using the MPEG-2 Standard.....	119
Table 11.5: ATM Classes of Service.....	121

1 Introduction

Recent years have seen tremendous changes in both computing and communication technologies. Not only has computing technology evolved from terminal- and mainframe-oriented systems toward client and server systems, software technology is migrating from procedural to distributed-object computing and the component-based paradigm. Communication technology, on the other hand, has progressed from low-speed shared networking technologies such as Ethernet and Token Ring to switched, dedicated networking technologies such as Fast and Gigabit-Ethernet and asynchronous transfer mode networks (ATM). In addition to being able to transmit information at a high speed, these networks also offer quality-of-service (QoS) guarantees. In particular, ATM networks are able to guarantee QoS properties such as bounds on throughput, delay, jitter, and reliability. In the transport and network layers of the communication protocol stacks, the TCP/IP protocol stack has come out on top against the OSI, DECnet, SNA, IPX and other protocol stacks.

As far as the performance of processors and peripherals such as storage systems, real-time audio and video codecs is concerned, end-system technology is available in the marketplaces at price/performance categories geared for the consumer and business markets. From the application perspective, there is a great demand for distributed applications with a high degree of interactivity and audio/video content. Typical application areas are distance learning and teletraining, desktop video-conferencing, business TV, entertainment, and electronic commerce.

All of these applications are of a distributed nature: either of a client/server type such as video on demand or of similar functionality such as video-conferencing systems. Common characteristics are, first, that they are distributed, having communication requirements which are extremely diverse and, second, that they show a strong demand for varying classes of quality of service. In fact, with respect to quality-of-service concerns, they share a need for QoS control and management. This thesis argues that, from an application programmer's perspective, applications with demand for real-time communication should be handled like any other distributed application.

1.1 Problem Statement

In the literature, plenty of research work has reported on providing end-to-end QoS guarantees in the communication layers and in the management of system resources. Until recently, work has seldom focused on consistent end-to-end QoS management architectures or on the seamless integration of these QoS control and management architectures in an object-oriented framework used to build distributed applications.

Initially, this thesis work is motivated by the above-mentioned recent technological changes on the hardware side as well as on the software side and the need to provide quality-of-service assurance for real-time components such as video streaming or conferencing as part of distributed applications. However, it is stimulated by the application area in which this work was undertaken: supporting an organization's workforce in its life-long learning process through distance learning, e.g. Web-based training (WBT) and teletraining. Additionally, this application area has proved to be a superior example of a field where the novel concepts of this work can be applied practically.

As presented in chapter 4, an analysis of these applications show that they are of an inherently distributed nature, are service-oriented, e.g. have a demand for additional services such as accounting, billing, and security services, and require controlling and managing of continuous media transfer. The requirements and constraints for the architectural approach are given by the class of distributed multimedia applications which are to be consistently supported. The framework has to support not only the content creation, storage, delivery, and reception process of multimedia training material but also the essential accounting and billing mechanisms. The content itself incorporates a great deal of audio, video, and animations in different levels of quality depending on

the delivery mechanism used. If we look at the products currently on the market and the work done in research, analysis shows that there is no quality-of-service management architecture available which is based on the distributed-object computing technology paradigm and is able to fulfil all of the following requirements at the same time:

- *Provide end-to-end QoS at different levels of abstraction*, i.e. at the user, application, system, and network level. For example, a user might select the quality of a video with a slider between excellent, good, and poor whereas the audio quality should have priority over the video in most applications. This kind of quality could then be related to a pricing scheme. At each level of abstraction, QoS has to be expressed differently and guaranteed end-to-end.
- *Enable mapping of QoS specifications from one level of abstraction to another*. The desired QoS at the user level has to be mapped seamlessly between the different levels of abstraction.
- *End-to-end QoS control and management of a multimedia transport system* capable of transporting audio and video with different quality levels encoded utilizing the most commonly used video compression formats such as MPEG-2, MPEG-1, H.263, RealG2, and Quicktime.
- *Control and management of multimedia source and sink objects*, e.g. hardware and software encoders and decoders with cameras, VCRs, hard disks, video/media servers, etc.
- *Seamless integration in a distributed object-based computing paradigm and ability to hide the complexity of lower-layer communication application programming interfaces (APIs)*. This approach enables application programmers to concentrate on building their applications without having the need to know the details of the underlying communication and distributed system. The nature of an application – whether it is stand-alone or distributed – should be transparent to an application programmer. This ensures the same level of abstraction at the application programming interface for both the control and management of the audio and video streams and for the distributed distance learning application.
- *Platform independence*. The architectural framework should support a heterogeneity of hardware and operating systems on the client and server sides, as well as heterogeneous communication systems between them, because this reflects the reality in larger organizations.

1.2 Research Goal

The objective of this thesis is to design and experimentally validate a quality-of-service management architecture which is based on the distributed-object computing paradigm, i.e. so that a distributed multimedia application can be viewed from an application programmer's perspective just as any other application without real-time requirements. The work comprises six steps:

- *Requirements analysis*. By selecting a teletraining and distance learning application as a superior representative for a distributed multimedia application, the problem space is identified and generalized requirements are extracted. It is shown that these requirements can be generally applied to most distributed multimedia applications.
- *Specification of an architectural framework* which is capable of supporting both heterogeneous computing and communication systems and managing QoS-capable multimedia devices and communication systems.
- *Identification of the services and protocols in the communication system* which are able to support QoS.
- *Mapping these services to the application programmer's context* by introducing the concept of a communication- and platform-independent QoS application programming interface.

- *Separation of the transport of multimedia streams from the management* of the sources, communication systems, and sinks of the streams, so that they can be handled as any other distributed application and managed through the same concepts.
- *Validation of the architecture* through the design, prototypical implementation, and test in the context of a distance-learning application.

This thesis work is founded on the following assumptions:

First, an end-to-end communication system providing QoS assurance and multicasting through well-defined application programming interfaces for both high- and low-speed media streams through asynchronous transfer mode (ATM) networks and Internet communication technology. Both ATM and Internet technology have to be considered, because ATM is still the only technology which is capable of providing QoS assurance for both high- and low-quality media streams and Internet technology because of its overall availability.

Second, a Common Object Request Broker Architecture (CORBA)-compliant ORB as middleware infrastructure for building the distributed object-oriented applications. As will be discussed in section 2.3, CORBA is the only distributed system which is capable of being applied in heterogeneous systems.

Third, a high-performance UNIX-based server system with real-time guarantees. With respect to the scalability, stability, and system performance, UNIX-based server systems are the preferred choice for systems which have to deliver high-volume data with real-time guarantees such as high-quality video.

Fourth, standard personal computer and workstation equipment with hardware and software support for the encoding and decoding of media streams.

Fifth, in general, to consider using commercial off-the-shelf technology wherever possible in order to avoid designing, using, and implementing costly proprietary solutions. Commercial off-the-shelf hardware and software will be used for encoders, decoders, video/media servers, cameras, VCRs, communication and distributed system, if available and manageable through open application programming interfaces.

1.3 Outline of the Dissertation

After the general introduction contained in this chapter, chapter 2 presents the terminology, principles, and basic concepts of QoS management and control, object-based distributed systems, and application programming interfaces. The current state of the art including de-jure and de-facto standards are set out in chapter 3. A detailed requirements analysis is undertaken in chapter 4. Chapter 5 discusses the multimedia device and stream management architecture as a framework for this work. Details of the stream control architecture and the management service are discussed in chapters 6 and 7. The implementation is presented in chapter 8. Experiments are described in chapter 9. Finally, conclusions are drawn, the summary of contributions made, and further research work is presented in chapter 10.

2 Terminology, Principles, and Basic Concepts

This chapter presents the terminology, principles, and basic concepts which are used in later discussions of quality of service, object-based distributed systems, and application programming interfaces and which are most relevant for this work. Where appropriate, the terminology defined by standardization groups such as ISO, IETF, ATM Forum and OMG is used. The focus of this work lies in the definition of a consistent framework for the control and management of multimedia transport streams through object-based distributed system technology, therefore only the relevant aspects are discussed in detail.

2.1 Multimedia and Communication Systems

In several publications, a multimedia system has been defined from a technical perspective [StRüRa90] [StHe91] [StNa95] [Ste99]:

A multimedia system is characterized by computer-controlled, integrated production, manipulation, presentation, storage, and communication of independent information, which is encoded at least through a continuous (time-dependent) and a discrete (time-independent) medium.

With respect to this work, multimedia will be used throughout the rest of this work as per this strict definition. Nevertheless it has to be noted that the term multimedia is frequently used in a much broader sense, where no continuous medium is present.

With this definition in mind, a multimedia communication system deals with the transfer, the protocols, the services, and the mechanisms of discrete media data such as text and graphics and of continuous media data such as audio and video in digital communication networks. The communication system as one of the major parts of a multimedia system has to be able to handle well-defined quality of service (QoS), as explained in depth in the next section, on an end-to-end basis. The most important QoS parameters are compliance with end-to-end delay and jitter requirements as timing restrictions and restrictions of the loss characteristics [WoGrSt97] [SchZi95]. They are typically used to request the required capacities of the resources concerned.

2.2 Quality of Service

Quality of service (QoS) is a general term that covers system performance, as opposed to system operation or function. A system is specified, designed, and built to perform a well-defined set of functions for its users. These operational or functional requirements of a system take time and require system resources to perform within the specified limits. Occasionally they are subject to system errors or failures. These and other similar features are performance or non-functional features of a system and are often summarized under the heading of quality of service (QoS).

Another approach to defining QoS is to identify the system properties of interest by means of examples. From this perspective, QoS can be said to be concerned with the properties of system functions and information such as delay, response time, throughput, stability or usability of continuous media output, the freshness and accuracy of information, the coherence of distributed information, availability, etc. A useful but somewhat simplified summary is to say that QoS is concerned with timeliness, accuracy, and integrity.

Since QoS is a pervasive requirement, the user's requirements for QoS are reflected in each and every part of a system from end to end. QoS involves actions in end systems, communications systems, and any other parts of any system through which end-to-end traffic may pass or where

interaction is required. Therefore, the user's requirements for QoS have to be translated or mapped into requirements for specific processing such as scheduling, technology, design, and implementation in order that they may be achieved.

Traditionally, QoS issues were considered during the design phase. Mostly, the system hardware and software were engineered in an extremely "static" fashion to meet QoS requirements. The design choices were often made on the basis of simulations and predictions of the performance and behavior of various system elements. Typically, this involved choices of hardware-processing capabilities, operating and system software, memory size, communication bandwidth, etc. These systems were often overengineered to be able to deliver acceptable performance even in worst-case scenarios. However, once these systems were operational, no further action was taken with regard to the QoS unless it worsened beyond acceptability.

This rather static approach to QoS is not good enough to meet many of today's system and user requirements, especially with respect to the scalability, flexibility, and fast introduction of new services. The extensive use of distributed and multimedia systems over shared networks for a variety of different and independent traffic streams and patterns is leading to the development of systems with built-in QoS manageability and controllability. Such systems are able to handle QoS requirements dynamically, negotiate QoS agreements and contracts, and then manage QoS through techniques such as mapping, monitoring, resource reservation, admission control, scaling, filtering, application adaptation, renegotiation, etc.

Thus, the notion of quality of service has evolved as a research topic in several fields of application. The term originally stems from research in communication systems to support multimedia applications which make use of continuous media such as audio and video. Soon after these early beginnings, it was recognized that QoS had to be provided not only in the communication system itself but also in various system layers [StWo97] [ChSyLa97]. In [StNa95], four layers of QoS are distinguished: user QoS, application QoS, system QoS, and network QoS. Even though the concept of QoS is quite different in each of these system layers, it has to be mapped seamlessly from one level of abstraction to the other. With respect to the communication system, a consensus has been reached that throughput, delay, jitter, and reliability are the main QoS parameters [WoGrSt97] [SchZi95].

The term quality of service has been defined by various authors, groups, and standardization bodies - all looking at it from their different viewpoints:

The most general definition of QoS is given in parts 2 and 5 of the "Basic Reference Model of Open Distributed Processing" (ODP) [ITU95, ITU97a]: *Quality of Service (QoS) - A set of qualities related to the collective behavior of one or more objects.*

The above-mentioned recommendation extends the management of QoS given in the ISO QoS framework standard [ITU97b] to the much wider context of distributed processing in general. The ISO QoS framework largely focuses on how QoS can be characterized, specified, and managed in communication systems.

The purpose of the QoS Green Paper prepared by the Object Management Group (OMG) [OMG97] was to define a set of necessary extensions to the object management architecture (OMA) and the CORBA standards that would support QoS in CORBA-based distributed-object systems. The definitions set out in the QoS Green Paper are based on those used in the ODP and ISO QoS frameworks cited above.

In [Ste99] on page 238, QoS is defined as follows: *Quality of service is characterized by the defined and controllable behavior of a system with respect to quantitative measurable parameters.*¹

More communication systems focused is the definition of QoS given in [ITG98]: *The totality of quality characteristics of a communication network from the viewpoint of the users of a considered service.*²

QoS within the ATM Forum traffic management specifications is defined as follows [ATMF96a] [Alles95] [ATMF95a] [ATMF96b] [ITU91]: *Quality of service is defined on an end-to-end basis in terms of the following attributes of the end-to-end ATM connection: cell loss ratio, cell transfer delay, cell delay variation.*

It is obvious from these definitions that QoS is, as pointed out at the beginning of this section, relative – and typically depends on the perspective from which it is viewed. Therefore, QoS has to be put in the user's application context and then specified from different perspectives and for each system entity.

As described in OMG's QoS Green Paper [OMG97], there is a close relationship between QoS and QoP (quality of protection, i.e. security). For example, the ISO QoS framework standard [ITU97b] table of QoS characteristics includes typical QoP characteristics such as integrity, reliability, authenticity or confidentiality in addition to time- and capacity-related QoS characteristics. In accordance with the OMG policy, this work is based on the assumption that QoS/QoP intertwining is not a fundamental design pattern. This means that while QoS and QoP requirements may be expressed at the same interfaces, the QoS and QoP functions are specified and implemented separately. However, the focus of this thesis is on QoS and not on QoP, i.e. security issues. Nevertheless, it is clear that QoP requirements as well as any steps taken to meet them may degrade and impact the QoS negotiation process. The significant advantage in keeping QoS and QoP separate is that systems which are solely QoS- or QoP-enabled or which can handle both QoS and QoP issues may then be built.

2.2.1 The Dimensions of the QoS Problems

QoS management is a complex challenge with a variety of dimensions. The major reason for the complexity of the problem is that QoS issues have to be handled on an end-to-end basis and that a variety of separate components are involved. Several of these dimensions are presented in this section:

- The best-effort-to-static-to-dynamic spectrum.
- QoS requirements, agreements, and contracts.
- Mapping of QoS requirements.
- Resource management, scaling, filtering, and application adaptation.
- Pricing.

¹ "Dienstgüte kennzeichnet das definierte, kontrollierbare Verhalten eines Systems bezüglich quantitativ meßbarer Parameter" [Ste99], page 238.

² "Gesamtheit der Qualitätsmerkmale eines Kommunikationsnetzes aus der Sicht der Benutzer eines betrachteten Dienstes" [ITG98].

2.2.2 The Best-Effort-to-Static-to-Dynamic Spectrum

Although QoS aspects are always taken into consideration in the design and implementation of a system, there is a wide spectrum of ways this can be accomplished. At the one extreme, we find the Internet's best-effort approach with no QoS guarantees at all. But even there, with a combination of overprovisioning of bandwidth, application adaptation, and flat-rate pricing, this approach still manages to satisfy the users' needs. More common is the static approach to QoS, in which QoS requirements are determined and specified during the system analysis and design process and are met through a combination of system design choices. In general, such design options consist of software configurations, e.g. scheduling algorithms, priority assignment to tasks, etc., and hardware configurations, e.g. memory capacity, circuit bandwidth, etc. Today's telephone systems are typical examples of static QoS solutions. At the other end of the spectrum are systems designed to operate with a range of QoS requirements whose elements dynamically negotiate within the range of the QoS required, initiate functions to maintain agreed QoS levels, monitor what is actually provided, and adapt as far as possible to any service degradations that cannot be prevented or remedied.

2.2.3 QoS Requirements, Agreements, and Contracts

In principle, users can express their requirements dynamically at the beginning of a session – either interactively or by means of preset profiles – and during a session. A QoS requirement is satisfied by the implicit or dynamic negotiation of QoS agreements or contracts. An agreement may be regarded as a common understanding between a service user and a service provider. Generally, a contract tends to be more binding than an agreement since it often covers more global content, which is then exactly stipulated in detail, naming the precise consequences if breached.

The provisions of a QoS agreement regarding a given QoS requirement need to include the following: the constraints on the QoS characteristics derivable from QoS requirements and the level of agreement, i.e. whether it is either a best-effort agreement or warranted with some estimated probability, or guaranteed, whether it is a legal contract between a service user and a service provider with a well-defined Service Level Agreement (SLA), which can be mapped into a technical QoS specification, and whether the QoS achieved is to be monitored and what action is to be taken if the requirement is not met temporarily or permanently.

In general, a QoS agreement will require some actions to be performed by the end-to-end system, e.g. system-internal mapping and, at system boundaries of QoS agreements, resource management, scaling, filtering, application adaptation, etc. as described in the following sections [WoGrSt97] [Wolf96] [FHBM97].

Whereas, in static systems, these agreements are reached as part of the analysis, design, and implementation processes, in dynamic systems, they are the result of the negotiation process.

2.2.4 Mapping of QoS Requirements

The translation of QoS parameters from user requirements into system parameters and from one level of abstraction to another level is often called QoS mapping or QoS translation [KoNa97]. The user requirements can be expressed either explicitly through the setting of application preferences or of a user profile or implicitly through features of application objects [OMRW97]. For example, user requirements for video streaming will turn out to impose requirements on system capacity, throughput, and latency; user requirements for accuracy will lead to requirements on the reliability of data transfer and, in the case of billing, perhaps to the need for transaction processing. As the application focus of this work is on distributed systems, choices have to be made as to the distribution of system functions between object-based platforms, e.g. implementations of CORBA or Java RMI, and the use of shared processing and communication resources. With distributed object-based platforms, functions such as QoS mapping, the search for an acceptable service provider, and the conclusion of an agreement can be delegated to mapping and trading services.

Usually, the initial set of user QoS requirements has to be mapped into additional QoS requirements in greater detail.

2.2.5 Resource Management, Scaling, Filtering, and Application Adaptation

Resource management is one way to try to provide the QoS desired [Wolf96] [Wolf98]. This can be done by measures such as resource reservation, admission control, and QoS routing, to name just a few. Media scaling and filtering [WoGrSt97] and application adaptation [OMRW97] [TaBrSm97] [Tas97] are all basically attempts to live with the currently available QoS and provide a degraded – but still tolerable – service to the users. This may be done in several ways: for example by dropping some MPEG frames or by changing the picture size in video or by sacrificing the accuracy of some information.

2.2.6 Pricing

Current pricing policies for existing networks are either flat-fee access-based or time- and/or volume-based. These rudimentary charging schemes are only suitable for single-service networks with coarse-grain service level agreements or in best-effort service networks such as the current Internet. In multi-service networks or in networks with resource reservation, i.e. QoS on demand, coarse-grain charging schemes are no longer sufficient. This has led to controversy on how pricing can be done in computer networks [SCEH96] and in conjunction with QoS-capable networks [FeDe98]. Cost and pricing schemes for both Internet Integrated Services (cf. section 3.1.4.2) [KSW99a] [KSW99b] and Internet Differentiated Services (cf. section 3.1.4.3) [SLCL99] are under discussion.

2.3 Object-based Distributed Systems

The concept of distributed-object technology fuses the best of object-oriented programming with the client-server paradigm: abstraction to reduce the complexity and make programming easier and distribution to rightsize hardware and software.

2.3.1 OMG OMA

The Object Management Group (OMG) is an international vendor-independent nonprofit association which aims at maximizing the portability, reusability, and interoperability of software [OMG00]. The Object Management Architecture Guide (OMAG) describes the conceptual framework upon which the supporting specifications are based. It includes the OMG object model, which is implementation independent and defines common semantics for standard specification of the externally visible characteristics of objects, and the OMA reference model with its interface categories as depicted in Figure 2.1.

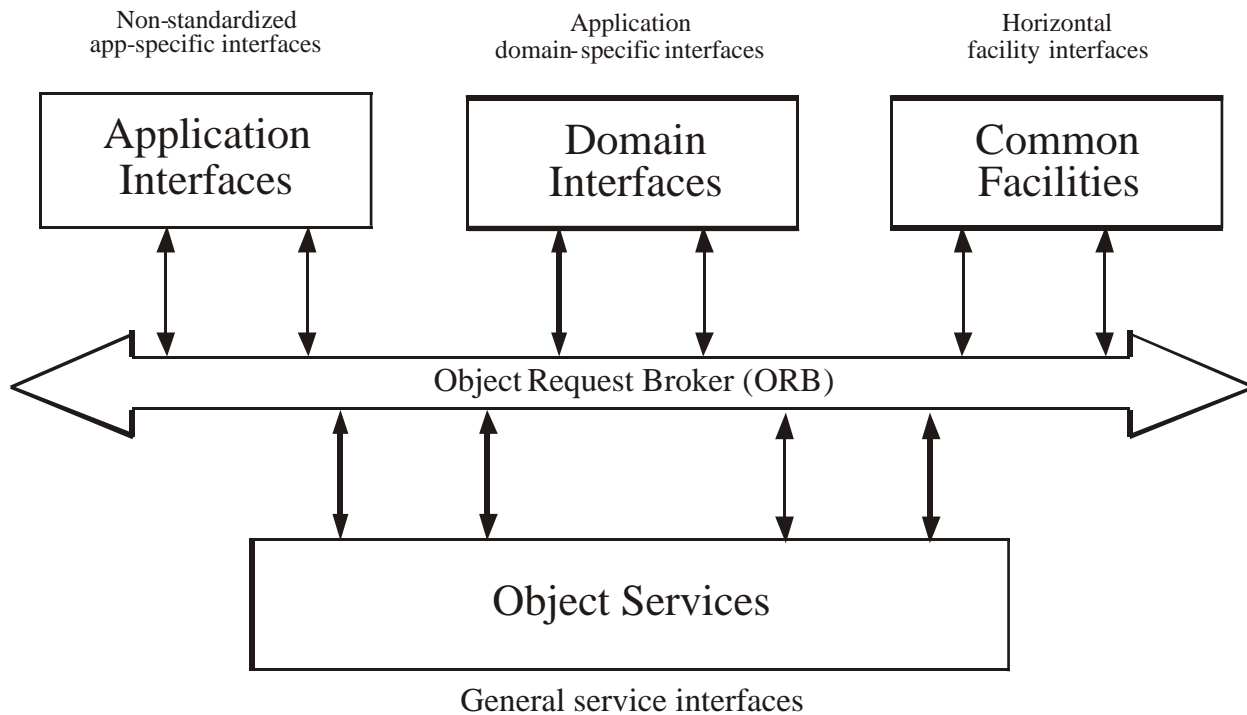


Figure 2.1: OMA Reference Model – Interface Categories

The reference model identifies and characterizes the components, interfaces, and protocols comprising the OMA. It consists of the object request broker (ORB) component, which enables objects to communicate in a distributed environment, and four interface categories:

- Object services (CORBA services) – widely available interfaces which are likely to be used in any object-based distributed system.
- Common facilities (CORBA facilities) – interfaces for horizontal end-user-oriented facilities applicable to most application domains.
- Domain interfaces – application domain-specific interfaces.
- Application interfaces – non-standardized application-specific interfaces.

The Common Object Request Broker Architecture (CORBA) defines the programming interfaces to the OMA ORB component. An ORB is the basic mechanism by which objects transparently make requests and receive responses from each other on the same or on different machines. With respect to client objects, the ORB transparently hides not only the mechanisms used to communicate with an object or to activate it, but also how the object is implemented and where the object is located. Therefore, the ORB is essential for building distributed applications constructed from distributed objects and for interoperability between applications in both heterogeneous and homogeneous environments [Vin97].

Unfortunately, there is currently no agreed and consistent way of specifying and managing QoS in CORBA-based systems [BeGe99].

2.3.2 Java RMI

With the release of JDK version 1.1, Java launched its own, built-in native ORB called RMI (remote method invocation). Even though RMI is an ORB in the generic sense, i.e. it supports making method invocations on remote objects, it is not a CORBA-compliant ORB. RMI is native to Java and is basically an extension of the core language. RMI depends on several of the other features of Java-object serialization: portable, down-loadable object implementations and Java

interface definitions. The resulting mechanism is very natural for Java programmers to use as they are never required to leave the Java programming environment or learn any new, “foreign” technology. On the other hand, RMI has some limitations with the major constraint being a consequence of its greatest strength: its tight integration with Java makes it impractical for use with objects or applications written in any other language.

The conventional use of the platform is described below:

The interface of an object which is to be made remotely accessible is specified in the form of a Java interface description. Parameters of methods are described by classes that implement the serializable interface. The interface specification of the remotely accessible objects is used by a compiler that generates additional classes, enabling a transparent method invocation from client to server. The handling of sockets and low-level network programming take place within some library classes and the generated code.

Serialization is a Java technology which enables the conversion of an instance of a class into a byte-stream and vice versa. In connection with RMI, this technology is used to transmit an arbitrary argument between the client and server of a method invocation. That means that any data structure to be used as an argument for an RMI method is a serializable class. In general, no additional coding is required to make a class serializable. It is a kind of a tag put on a class; the rest is done by the JDK.

A simple naming service – the `rmiregistry` – is provided with Java RMI for the identification and registration of server objects. The `rmiregistry` contains names and references to local server objects. A client establishes a connection to a remote object via the `rmiregistry`. At present, the `rmiregistry` only provides a transient naming service, which means that object names and references are only available from the registry as long as the process is running. After a shutdown of the registry, the server objects have to re-register.

As a rule, Java RMI uses a proprietary protocol (JRMP) for the communication between client and server. From JDK 1.2, the standardized IIOP (Internet Inter ORB Protocol) from the CORBA specification is available (RMI over IIOP) as an alternative. But some restrictions apply when using RMI over IIOP. Java RMI does not provide any means of specifying QoS issues.

2.3.3 DCOM

DCOM [CHY+98] is the distributed extension of COM (component object model) [Brock95], which builds an object remote-procedure call (ORPC) layer on top of DCE RPC to support remote objects. A COM server can create object instances of multiple-object classes. A COM object can support multiple interfaces, each representing a different view or behavior of the object. An interface consists of a set of functionally related methods, and a COM client interacts with a COM object by acquiring a pointer to one of the object's interfaces and invoking methods through that pointer, just as if the object resided in the client's address space. COM specifies that any interface must follow a standard memory layout – which is the same as the C++ virtual function table. Since the specification is at the binary level, it permits integration of binary components which may be written in different programming languages such as C++, Java, and Visual Basic. DCOM is a proprietary technology developed by Microsoft, and, similar to CORBA and Java RMI, DCOM does not provide any QoS support.

2.3.4 Comparison of Distributed-Object Technology

Table 2.1 gives an overview of the differences between CORBA and Java RMI [OrHa97] [Curtis97]:

CORBA	Java RMI
standardized	Java specific
open communication protocol	IIOP since JDK 1.2
activation component	-
dynamic invocation/skeleton interface	introspection / -
interface repository	- / introspection/class loader
platform- and language-independent	runs on top of Java platform
embedded in architectural framework (OMA)	-
standardized services	some Java-specific services
programmer has to learn IDL-mapping	Java-only, simple
different parameter semantics (in/out/inout)	call-by-value only
one-way methods	-
deferred synchronous calls (DII)	-
object references or data structures as parameters	object references or objects as parameters

Table 2.1: Functional Comparison of CORBA and RMI

As the overview illustrates, in many areas CORBA is more powerful than Java RMI: it is standardized, provides both an activation component and additional services, and it supports different programming languages. Although the dynamic components of CORBA, e.g. DII, DSI, and Interface Repository, seem to provide enhanced functionality compared to JAVA at the first glimpse, Java's introspection mechanism allows an object to analyze its own interface. Together with the Java dynamic class loader, this results in a similar functionality to the DII/Interface Repository in CORBA.

Another benefit of CORBA is the fact that the programmer can declare a method to be “one-way”. This may be done for methods without return values. Calling the method is non-blocking, i.e. the client process can continue and does not have to wait for an answer from the server process. The disadvantage of one-way methods lies in the fact that the method cannot have any return values. When using the dynamic invocation interface, the programmer may use so-called deferred synchronous calls, i.e. the client of a method call sends a request to a server, continues without blocking, and can later check for and retrieve the results and return values of the call. The characteristics of non-blocking method calls are especially valuable in a distributed environment with a long round-trip delay for network messages such as with a WAN connection on the Internet

or connections via mobile network services like GSM (Global System for Mobile Communications).

The major advantage of Java RMI is that it is easy to use, as it is a Java-only solution. Another valuable benefit is that the serialization mechanism permits whole objects to be transmitted as parameters. This enables transmission of the behavior, i.e. the methods of the objects, thus providing an agent-like functionality for free. In addition, subclasses of formal parameters of a method can be provided in calls.

2.4 Application Programming Interfaces

An API provides access to the services of the operating system. It hides the complex functionalities and implementation details from the application programmer. APIs can be found at different abstraction levels. As discussed in the previous section, high-level APIs such as CORBA, DCOM, and RMI [Curtis97] [OrHa97] are used to build applications based on the distributed-object computing paradigm. Basic-level APIs can typically be found at the boundary between user and kernel space. The BSD socket interface used in UNIX and in Microsoft Winsock is a typical example. Low-level APIs provide access to each of the different communication layers, e.g. the transport, network or data link layers. These APIs can be used to migrate the QoS features of the networks to the applications.

Realistically, there are currently only three approaches which have a chance of finding broader acceptance. The first approach is the Resource Reservation Protocol (RSVP) [BZBHJ97], which promises to provide a receiver-initiated setup of resource reservations for multicast or unicast data flows. The second approach is the so-called differentiated services approach. The third approach comprises using the built-in QoS features of ATM and delivering these native ATM services to the application. Combinations of both approaches, e.g. RSVP on top of native ATM, are also under discussion [BrSt97]. Therefore, these approaches will be discussed in more detail in chapter 3.

2.5 Video Compression

Digitalization of video data leads to an extremely large data flow if the data stream is uncompressed. For example, video data in the CCIR 601 standard (D1 standard) generates a data stream of 140 Mbps. In contrast to the transmission of video data over broadband networks (e.g. ATM networks with a transfer rate of 155 Mbps), transmission of such data via networks operating in the mid-bandwidths (up to approx. 15 Mbps) has to be performed using compression methods to reduce the amount of data transmitted.

Conventional, standardized data compression methods are lossy and exploit the physiological characteristics of the human eye (e.g. subsampling of chrominance values of images) and the attributes of the image to be reproduced (frequency analysis, motion prediction). Moreover, these are hybrid methods containing a variety of concepts for data compression (e.g. entropy coding), thus enabling a compression rate of 100:1. If the transmission bandwidth is sufficiently wide however, the methods utilized may scale down to a lower compression rate (adaption), thereby allowing the user to exert some degree of influence on the quality of the data transmission. In addition to the internationally standardized methods used, e.g. the H.261, MPEG-1, and MPEG-2 standards, proprietary standards such as RealVideo or Quicktime, which are not discussed in this work, must be mentioned. Also, for reasons pertaining to their characteristics – for example, resolution and bandwidth requirements – M-JPEG and AVI coding are not taken into account here.

A more detailed overview of the video compression techniques and standards such as H.261, MPEG-1 or MPEG-2 used in the prototypical implementation of this thesis (cf. chapter 9) is given in appendix 11.1 and in [EfSt98].

2.6 Summary

In this chapter, the significance and the various perspectives of QoS control and management in distributed multimedia systems has been indicated. The essential notions of multimedia systems, multimedia communication, object-based distributed systems, application programming interfaces, and video compression techniques have been defined and introduced. They are the fundamental building blocks for a multimedia device and stream management architecture, which will enable the creation of distributed multimedia applications just as any other distributed application.

3 State of the Art and Related Work

This chapter assesses the state of the art in quality-of-service standards and research. First, work in standardization bodies such as ISO and ITU-T is presented and discussed. Next, work carried out in forums such as the IETF, the ATM Forum, and the OMG is described. And, finally, work done in various research groups is examined – all of which represents the state of the art with regard to different parts of an end-to-end system such as the transport and network layers, operating systems, distributed system platforms, and integrated QoS management approaches.

3.1 QoS in Standards

For the end-to-end provisioning of QoS, it is crucial that de-jure or de-facto standards be used as extensively as possible. And even more important is that these standards be implemented – or at least have a chance of being widely deployed – in the future. So, even though the OSI reference model QoS features mentioned below are only of a theoretical nature, they are useful for clarifying which QoS parameters should be covered by an implementation. Some of the work done within the IETF is still in the early stages of development, e.g. DiffServ working documents have not yet been ratified as accepted standards.

3.1.1 ISO Open Systems Interconnection

One of the first attempts to standardize QoS issues was made by the International Standardization Organization (ISO). ISO has developed a set of standards for computer communications in accordance with the seven-layer OSI (open system interconnection) basic reference model [ISO84]. The OSI basic reference model is still a widely accepted conceptual framework. However, only a few of the OSI standards are in practical use today. The QoS support provided by the OSI reference model reflects the limited QoS requirements of data-only applications running over low-speed networks.

Table 3.1 lists OSI performance-oriented QoS parameters and Table 3.2 those QoS parameters which are non-performance oriented. The latter take protection-, priority-, and cost-related QoS categories into account. The performance-oriented QoS parameters are handled in the transport layer, and the QoS parameters relate to the phases of a session: connection establishment, data transfer, and connection release. Although these QoS parameters are not used in practice, they build the foundation of later research and are, thus, mentioned for this reason.

Parameter	Description
Throughput	The maximum number of bytes contained in service data units (SDUs) which may successfully be transferred in unit time by the service provider over the connection on a sustained basis.
Transit delay	The time delay between the issuing of a data.request and the corresponding data.indication. The parameter is usually specified as a pair of values: a statistical average and a maximum. Those data transfers where a receiving service user exercises flow control are excluded. The computations are all based on SDUs of a fixed size.
Residual error rate	The probability that an SDU is transferred with error or that it is lost or that a duplicate copy is transferred.
Establishment delay	The delay between the issuing connect.request and the corresponding connect.confirm.
Establishment failure probability	The probability that a requested connection will not be established within the specified maximum acceptable establishment delay as a consequence of actions that are solely attributable to the service provider.
Transfer failure probability	The probability that the observed performance with respect to transit delay, residual error rate or throughput will be worse than the specified level of performance. The failure probability is, as such, specified for each measure of performance of data transfer discussed above.
Resilience	The probability that a service provider will, on its own, release the connection or reset it within a specified interval of time.
Release delay	The maximum delay between the issuing of a disconnect.request primitive by the service user and a corresponding disconnect.indication primitive issued by the service provider.
Release failure probability	The probability that the service provider will be unable to release the connection within a specified maximum release delay.

Table 3.1: OSI Performance-oriented QoS Parameters

Parameter	Description
Protection	The extent to which a service provider attempts to prevent unauthorized monitoring or manipulation of user data. The level of protection is specified qualitatively by selecting either (i) no protection, (ii) protection against passive monitoring, (iii) protection against modification, addition or deletion, or (iv) a combination of (i) and (ii).
Priority	High-priority connections are serviced before those of lower priority. Lower-priority connection packets are dropped before high-priority packets should the network become congested.
Cost determinants	A parameter which defines the maximum acceptable cost for a network connection. It may be stated in relative or absolute terms. Final actions on this parameter are left to the specific network providers.

Table 3.2: OSI Non-Performance-oriented QoS Parameters

3.1.2 ITU-T and ATM Forum

This section describes the state of the art of the asynchronous transfer mode (ATM) technology with its inherent QoS features. ATM was standardized by ITU-T and the ATM Forum. Furthermore, ATM-aware communication APIs, which enable application programmers to use these QoS features, are outlined.

ATM-Layer Quality of Service

The ATM-layer quality of service (QoS) is measured by a set of parameters characterizing the performance of an ATM-layer connection [KKR97]. These QoS parameters quantify end-to-end network performance in the ATM layer. In ITU-T Recommendation I.356, they are referred to as network performance parameters.

Quality-of-Service Parameters

Six QoS parameters, which correspond to a network performance objective, are identified in this specification. Three of these may be negotiated between the end systems and the networks. One or more of the values of the QoS parameters may be offered on a per-connection basis, corresponding to the number of related performance objectives supported by the network. Support of different performance objectives can be done by routing the connection to meet different objectives or by implementation-specific mechanisms within individual network elements.

The following QoS parameters are negotiated: peak-to-peak cell delay variation (peak-to-peak CDV), maximum cell transfer delay (maxCTD), and cell loss ratio (CLR). The following QoS parameters are not negotiated: cell error ratio (CER), severely errored cell block ratio (SECBR), and cell misinsertion rate (CMR).

The classes of service of ATM are shown in Table 11.5 of the appendix. Further information on ATM-layer QoS may be found in ITU-T Recommendation I.356.

ATM-aware Communication APIs and ATM QoS APIs

In order to advance the development of native ATM services, the ATM Forum has released the first version of a semantic description [ATMF96d], which is to provide an engineering foundation for the development of APIs that make use of ATM-specific services.

Native ATM services include data transfer – including both reliable and unreliable data delivery – using the ATM layer and various ATM adaptation layers, provisions for setting up switched virtual circuits (SVC) and permanent virtual circuits (PVC), traffic management considerations – including traffic types and quality of service guarantees, distribution of connections and associated data to the correct application or entity, and provisions for local participation in network management.

The reference model for native ATM services developed by the ATM Forum shown in Figure 3.1 distinguishes between native ATM APIs on the left and existing transport APIs on the right-hand side of the figure.

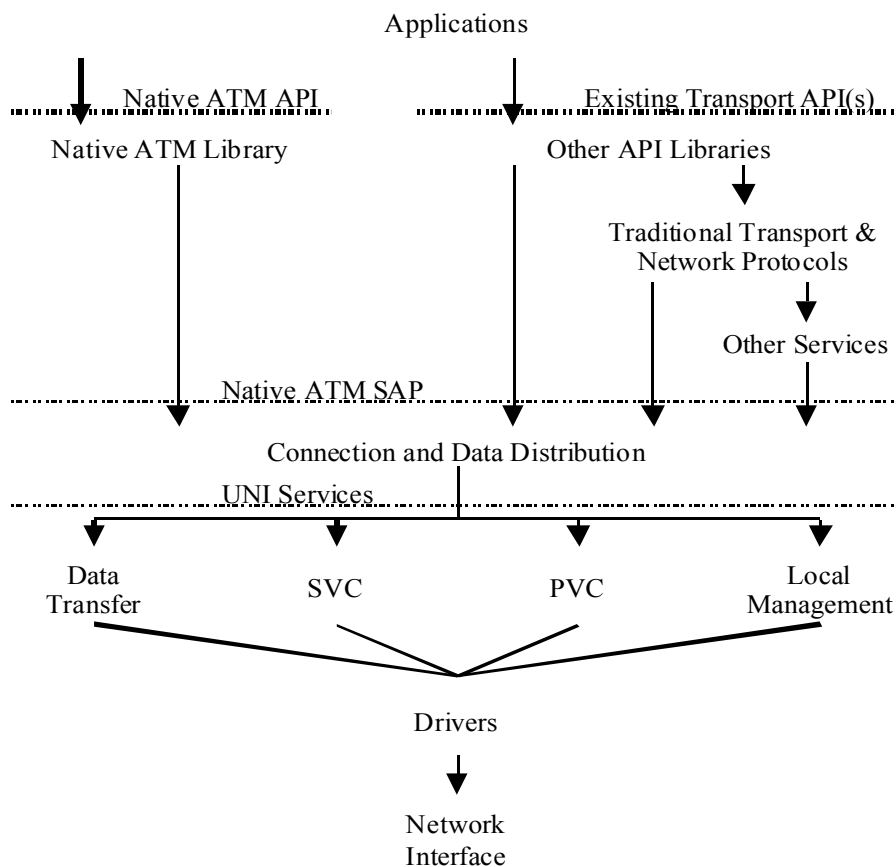


Figure 3.1: Reference Model for Native ATM [ATMF96d]

Native ATM APIs

Native ATM APIs make direct use of the native ATM service access point. They are independent of constraints of the operating system or of existing communication APIs. The application has to use the syntax and semantics of the specific API, thus restricting the application to the related API. Typical examples are the proprietary APIs of ATM boards, which are marketed by a number of vendors to support their products.

The use of native ATM APIs is sensible in areas where only a restricted set of the ATM functionality is required. Application areas are telephony, videotelephony/video conferencing or distance learning applications. For all of these applications, APIs with an abstraction of implementation details of the lower layers make sense. With the above-mentioned release of a semantic specification for native ATM services, the ATM Forum is aiming at providing a foundation for native ATM APIs.

Existing Transport APIs

The second group comprises extensions of existing transport APIs. As a rule, these APIs are a part of the operating systems. Using the reference model for native ATM APIs and the semantic description, the ATM Forum and vendors have extended the traditional interfaces of UNIX derivatives, Microsoft Windows 95/98/NT, and Apple MacOS.

In particular, the existing APIs are being enhanced with the ATM capabilities of UNI 3.0 and 3.1. All of the APIs support 1:N multipoint connections. In addition, the leaf-initiated join multicast to an existing multicast session will be provided. Existing APIs support either reliable (SSCOP, service-specific connection-oriented protocol) or unreliable data delivery. Currently, only AAL 5 (ATM adaptation layer) is supported in the message mode, which means that each AAL 5 service data unit is mapped into the AAL 5 payload field. So far, the following specifications have been published:

- The traditional UNIX socket interface supports AAL 5 [ATMF96e].
- The Open Group API X/open transport interface (XTI) [ATMF96e] only supports AAL 5.
- The data link layer provider interface (DLPI) [ATMF97e] supports AAL 5, 1 and a user-definable AAL. Of AAL 1 and the user-definable AAL, only the AAL 5-based signaling part is supported. The data transmission part still remains an issue for further study.
- Apple has integrated XTI and DLPI into its open transport (OT) specification and implementation.
- Independently of the ATM Forum, Microsoft has developed the Winsock 2 specification with its own syntax [ATMF96f]. Essentially, it is derived from the Berkeley System Distribution (BSD) socket syntax. The syntax mapping is described in [ATMF96c].

3.1.3 UNIX ATM API

The native ATM service specification for XTI and DLPI enables transparent access to the UNI 3.x services in UNIX environments. Initial implementations of the specification have been made available by vendors of ATM adapters. In Figure 3.2, the UNIX ATM architecture which deploys XTI, XSocket, and DLPI ATM APIs is depicted.

Access to the ATM services is provided either via the XTI/Xsockets or via DLPI, which permits the use of data link layer services. The support of several networking technologies is planned or has already been integrated within DLPI version 2.0 [OG97a]. In addition, DLPI is the API of choice for LANE and Classical IP.

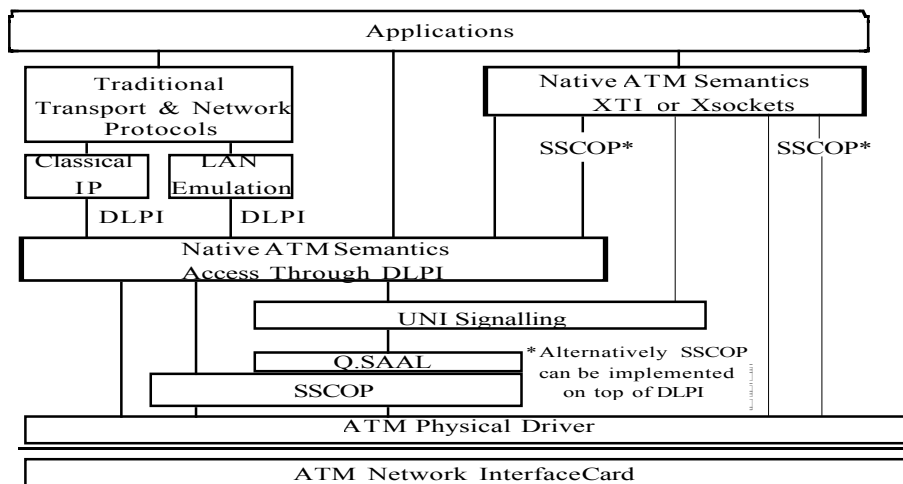


Figure 3.2: UNIX ATM Architecture Using XTI, XSocket, and DLPI ATM APIs [ATMF97e]

The Open Group formed in 1996 by the merger of X/Open and the Open Software Foundation (OSF) is a vendor-neutral, international consortium for buyers and suppliers of technology with more than 200 member companies. The Networking Services (XNS) [OG97b] specification describes two APIs to transport-level process-to-process communications: sockets and XTI. One of the features of XNS includes the use of sockets and XTI in conjunction with ATM. The ATM transport protocol information for XTI is described in appendix L of the XNS specification. It corresponds to a preliminary specification and addresses an emerging area of technology. Hence, it is not yet supported by multiple sources of stable conformant implementations.

The preliminary specification supports only a subset of the functions specified by the ATM Forum in the User-Network Interface specifications versions 3.0 and 3.1[ATMF95a]. The ATM transport provider supports both reliable and unreliable connection-oriented transport services. However, the features listed are not available in the ATM transport provider for XTI: connectionless transport service, orderly release mechanism with `t_sndrel()` and `t_rcvrel()`, expedited data transfer, transmission of user data during connection setup and release, permanent virtual circuits (PVCs), congestion indication bit and user-user byte in AAL 5, AAL 1 and user-defined AAL, and alternate BLI (broadband low layer information) negotiation.

The signaling-level option `T_ATM_TRAFFIC` enables the definition of the data traffic descriptor with forward and backward peak cell rate (PCR), sustainable cell rate (SCR), and the maximum burst size (MBS). The `T_ATM_QOS` option is used to signal the quality of service desired (classes 0, 1, 2, 3 or 4).

Arequipa

[AlSaWr] describes Arequipa (Application REQuested IP over ATM) as a mechanism for the establishment of end-to-end ATM connections under the control of the applications and for the use of these connections with given QoS at the lower protocol layers to carry the IP traffic of specific sockets. Arequipa provides a native ATM service API for Linux.

ATM Forum SAA/API Working Group

The sub-working group SAA (Service Aspects and Applications)/API of the ATM Forum started in February 1998 with an outline for a working document to describe the functional primitives, states, and procedures for a Java ATM API [ATMF98a]. The intention of the document is to propose two distinct implementations: 1) a native ATM API which provides developers with direct access to ATM functions and 2) an API which is derived from the sockets API used to provide TCP/IP communications for Java programs.

The proposal for a native ATM API for Java is based on the ATM Forum contributions *A Java Syntax Instantiation of the Native ATM Services* [ATMF97c], *A Java Syntax Mapping to the Native ATM Services* [ATMF97d], and *A Java Syntax Implementation of the Native ATM Services* [ATMF97b]. Some advantages of this approach include the following: the ability to explicitly specify quality-of-service (QoS) parameters and flexibility in interworking with multimedia data formats.

The drawbacks of this approach are that it has nothing in common with the java.net package of the current JDK distributions.

The second approach for an ATM API for Java is founded on the existing Java implementation of Transmission Control Program/Internet Protocol (TCP/IP) as proposed in [ATMF97a]. The benefits of this approach are as follows: a TCP-based API will enforce similarities between TCP/IP and ATM for equivalent behaviors, and only minimal changes to existing programs using TCP/IP are required.

3.1.4 IETF

The Internet architecture is based on a packet-switched datagram network with IP as the glue between the applications and the transport protocols they are using and a variety of communication systems underneath IP. The Internet architecture is based on the stateless approach, i.e. there is no per-flow state information available inside the network. IP itself does not participate in resource management, accordingly providing only best-effort service, i.e. no quality-of-service guarantees. The key idea in the Internet architectural approach is to keep the network layer as simple as possible in order to support a variety of applications above it and a variety of data link and physical layer technologies such as Ethernet, ATM, modem, satellite or wireless below it. IP provides minimal functionality to support end-to-end connectivity such as addressing, forwarding, and routing. This is the key to the success of the Internet architecture over other networking technologies. The network itself is basically dumb, and the end systems have to be smart enough to perform more sophisticated functionalities such as flow, error, and congestion control. In contrast to IP, the telephony network is based on a highly sophisticated network with simple end systems.

With multicasting and multimedia applications in mind, the first QoS discussion started in 1989 in the IETF (Internet Engineering Task Force) end-to-end research group. In 1994, the IETF working groups on an Integrated Internet Services Architecture (IntServ) and RSVP (Resource Reservation Protocol) were formed. In 1997, as a consequence of the problems incurred in the IntServ and RSVP approach, the IntServ working group discussed another proposal for providing QoS in the Internet the so-called Differentiated Services Architecture, also known as DiffServ. This led to the formation of the DiffServ working group in February 1998. The next section is devoted to the state of the discussion with regard to traffic management in today's Internet, followed by the IntServ and DiffServ architectures. In fact, QoS policies are under discussion in the IETF, and multicasting is a long-term hot topic and crucial element in multimedia communication.

3.1.4.1 Internet Traffic Management

The Internet is a packet-switched network. The network resources are statistically shared between the end systems. Overloading the network causes congestion, resulting in packets being either delayed or dropped. The consequence is degradation in application performance. The challenge is to provide high resource utilization, at the same time preserving high application performance.

Congestion control in today's Internet is done by TCP as an end-system-only solution. The assumption is that users will cooperate voluntarily, and the policy is equal sharing of pain in the case of overload situations. TCP estimates the network state dynamically, so that when loss of packets signals network congestion, TCP reduces the transmission rate accordingly.

Currently deployed routers play only a marginal role in congestion and resource management. The router mechanisms which impact congestion management are scheduling and buffer management. Conventional routers use FIFO schedulers and tail drop. But these have some drawbacks: buffer lock-out due to misbehaving flows, a synchronizing effect for multiple TCP flows, and burst or multiple consecutive packet drops. Because all three behaviors are extremely poor for TCP fast recovery in case of congestion, several suggestions have been made for other scheduling and buffer management algorithms. For example, Cisco Systems has implemented the RED (Random Early Discard) [FlJa93] algorithm in its family of routers. RED is based on an estimation of the average queue length, and the discard probability is a function of the average queue length. The advantages of RED are that it can absorb burst better, avoids synchronization of multiple TCP flows, and signals a congestion to the end systems earlier.

The limitations of the resource management and congestion control deployed in the Internet at present are that it works only if most of the sources implement TCP and are cooperative. Basically, cooperative means that the end systems do not use variations of the standard TCP protocol. So applications such as audio/video streaming applications, which use UDP instead of TCP, are misbehaving applications and can grab most of the bandwidth of a link. Of course, the most eminent limitation is that the Internet cannot provide service assurance and service differentiation, i.e. no QoS.

This has led to the development of two different architectures which are now under discussion in the IETF and the research community: IntServ and DiffServ. These architectures represent two dimensions of QoS. The IntServ approach follows the flow-based end-to-end absolute performance assurance paradigm, regardless of the behaviors of other traffic. The DiffServ approach represents the class-based paradigm, where traffic is aggregated with the provision of relative QoS. Relative QoS means that QoS is defined with respect to other flows, e.g. priority or weighted fair share.

3.1.4.2 Integrated Internet Services Architecture (IntServ)

The goal of the Integrated Services Architecture (IntServ) [BrClSh94] is to enhance IP's best-effort single-service class model by a new model with multiple service classes, including best-effort and QoS classes. It introduces explicit resource management at the IP level and pioneers per-flow state which has to be maintained by the routers, thus showing a key difference to the previous model. The per-flow state is used for admission control and scheduling, while set-up and tear-down of the reservation state is done by RSVP [BZBHJ97]. The newly introduced service models are end-to-end per flow guaranteed [ShPaGu97], controlled load [Wro97b], best-effort, and hierarchical link-sharing. As depicted in Figure 3.3, the mechanisms used are admission control to determine if there are enough resources available and policy control to determine if the reservation is in accordance with preset policy rules. Traffic control tasks are twofold: first, to classify a packet to a flow and, second, to schedule packet transmission according to the per-flow state.

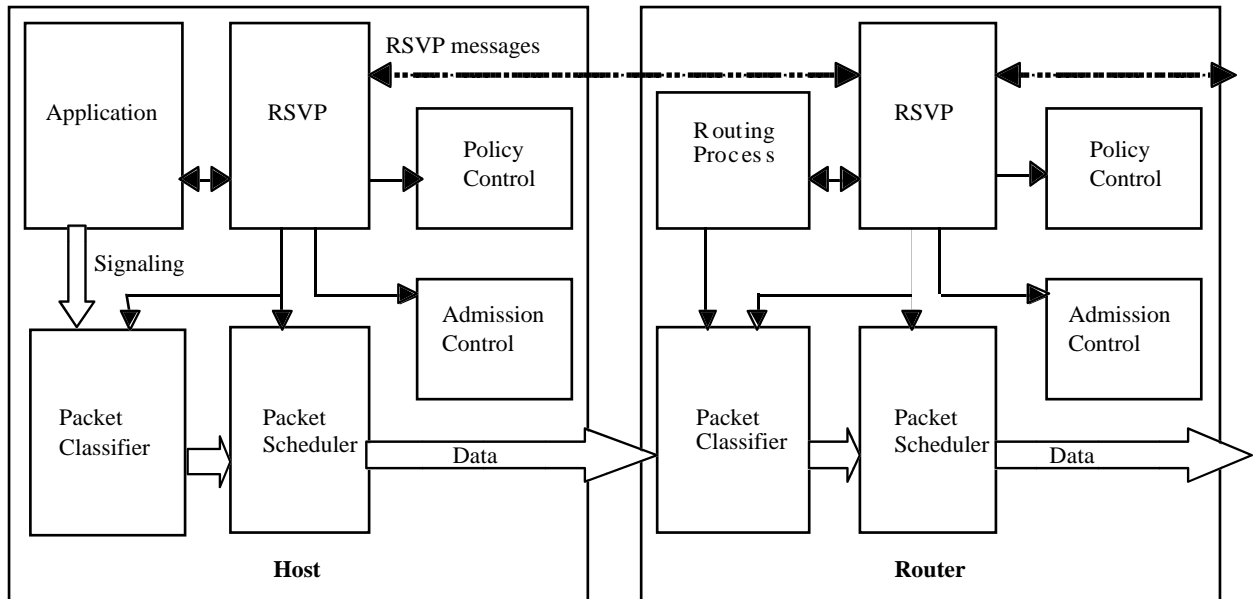


Figure 3.3: IntServ Protocols and Mechanisms

The guaranteed service intends to provide a hard real-time service. It guarantees a deterministic upper bound on delay for each packet in a session, under the provision that the session does not send more than it specifies. Admission control is based on worst-case analysis. Per-flow queuing has to be supported by the routers. In contrast, the controlled load service provides soft real-time service: a flow with a QoS closely approximating the QoS that the same flow would receive from an unloaded network. Measurement-based admission control and scheduling for aggregated flows are possible.

The hierarchical link-sharing service model is used for resource contention and sharing between different organizations such as the various departments of a university at different levels. Resource management policies should be set at the various levels by different entities such as resource owners, service providers, organizations, and applications.

The role of RSVP in the IntServ architecture is that of a signaling protocol for establishing per-flow state. It has to carry requests for resources from host to routers and collect any information needed from the routers to deliver to the hosts. At flow set-up, RSVP first has to consult admission and policy control. It then sets up the admission and flow state or informs the requester of the failure. RSVP's design features are its IP multicast-centric design, receiver-initiated reservation, and support of different reservation styles. In addition, RSVP establishes soft state inside the network and decouples routing from reservation signaling.

During the past three years, several problems have been identified in conjunction with IntServ and its core element RSVP. This resulted in an intensive discussion in both the IETF and the research community with no final conclusion so far reached. First, IntServ does not fit in the current Internet business model, where an end-to-end flow typically has to traverse the domains of several Internet Service Providers (ISPs). Second, it violates the critical success factors of IP – its simplicity and scalability. Because of the introduction of a potential high overhead of per-flow scheduling and buffer management, the maintaining of per-flow state, the processing of RSVP's signaling packets, soft-state refreshing, and timeout, IntServ's scalability is disputable. Also, due to RSVP's inherent scalability problems, the current consensus is that RSVP should be used only in access networks and, perhaps, in Intranets and that it would be preferable to use DiffServ in the core network as described in the following section.

The IntServ working group has undertaken to define the interfaces that express the application's end-to-end requirements, router scheduling interfaces that define the information to be made available to individual routers within the network, and (general) subnet interfaces.

Figure 3.4 gives a simplified overview of how RSVP works:

Senders characterize outgoing traffic in terms of the upper and lower bounds of bandwidth, delay, and jitter. RSVP sends PATH messages from the sender that contains this traffic specification (TSpec) information to the (unicast or multicast receiver(s)) destination address. Each RSVP-enabled router along the downstream route establishes a path-state that includes the previous source address of the PATH message (i.e. the next hop upstream towards the sender). To make a resource reservation, receivers send a RESV (reservation request) message upstream to the (local) source of the PATH message. In addition to the TSpec, the RESV message includes the QoS level required (controlled load or guaranteed) in an RSpec and characterizes the packets for which the reservation is being made (e.g. the transport protocol and port number), which is called the filter-spec. Together, the RSpec and filter-spec represent the flow-descriptor that routers use to identify reservations. When an RSVP router receives a RESV message, it uses the admission control process to authenticate the request and allocate the necessary resources. If the request cannot be satisfied (due to a lack of resources or an authorization failure), the router sends an error back to the receiver. If the request is accepted, the router sends the RESV upstream to the next router. When the last router receives the RESV and accepts the request, it sends a confirmation message back to the receiver (note: the last router is either closest to the sender or at a reservation merge point for multicast flows). There is an explicit tear-down process for a reservation when the sender or receiver ends an RSVP session.

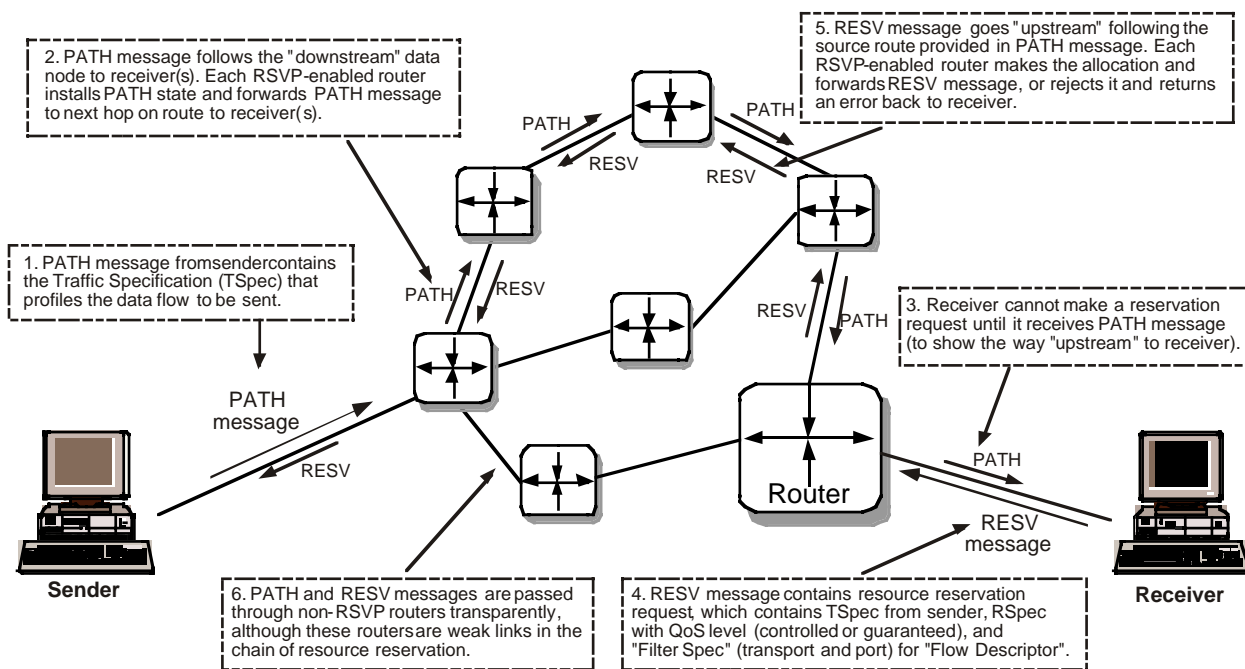


Figure 3.4: RSVP

Here are some salient characteristics of RSVP support:

Reservations in each router are soft, which means that they need to be refreshed periodically by the receiver(s). RSVP is not a transport, but a network (control) protocol. As such, it does not carry data; instead it works in "parallel" with TCP or UDP data flows. Applications require APIs to specify the flow requirements, initiate the reservation request, and receive notification of reservation success or failure after the initial request and throughout a session. To be useful, these APIs also need to include RSVP error information to describe a failure. Multicast reservations are merged at traffic replication points on their way upstream, which involves complex algorithms that are not yet

well understood. Although RSVP traffic can traverse non-RSVP routers, this creates a weak link in the QoS chain where the service falls back to best effort (i.e. there is no resource allocation across these links).

In summary, RSVP has the following characteristics:

- Resource reservation for both unicast and many-to-many multicast applications.
- Reservations for unidirectional data flows.
- Receiver oriented.
- Soft state.
- Depends upon present and future routing protocols.
- Transports and maintains traffic and policy control parameters.

3.1.4.3 Internet Differentiated Services Architecture (DiffServ)

The differentiated services approach to IP quality of service has its roots in several years of research discussions and ad hoc traffic engineering. The IETF working group on differentiated services, which was formed in early 1998, has produced three RFCs and an informational RFC on the general architecture (cf. section 11.3). A number of vendors have announced DiffServ products, and the Internet 2 effort has adopted a DiffServ-style QoS for its testbed. The initial momentum for DiffServ came largely from Internet service providers (ISPs) and reflected a lack of confidence in the IntServ/RSVP QoS solution that came out of the research community. DiffServ reflects the need for a scalable, incrementally deployable QoS that meshes well with the administrative realities of today's Internet.

The goals of the architecture framework are to provide service differentiation that is (1) highly scalable and relatively simple to implement, (2) applicable to intra-domain and inter-domain resource and QoS management, and (3) within the context of current organizational requirements and customer-provider business models. In order to achieve scaling, the following measurements are undertaken: (1) per-connection state is pushed to edge devices of the core network, (2) per-contract state is pushed to business boundaries, (3) internal nodes differentiate among a small number of traffic aggregates.

DiffServ was proposed to provide service differentiation by creating service classes with different priorities using either the type-of-service (TOS) field of IPv4 or the priority bits of IPv6 headers. This priority scheme translates into higher throughput for higher priority classes. The goal for differentiated services is to define a scalable service discrimination without the need for per-flow state and signaling at every hop.

DiffServ, as per RFC 2475 and 2474, defines a scalable service discrimination policy without maintaining the state of each flow and signaling at every hop. The primary goal of differentiated services is to permit different classes of service to be provided for traffic streams on a common network infrastructure. DiffServ aggregates a multitude of QoS-enabled flows into a small number of aggregates with these aggregated flows given differentiated treatment within the network. The DiffServ approach attempts to push per-flow complexity away from the network core and towards the edge of the network where both the forwarding speeds and the fan-in of flows are smaller. Each DiffServ flow is policed and marked according to the service profile at the edge of the network and services only a small number of traffic aggregates in the core.

DiffServ will provide a controlled and coarsely predictable IP class of service. To support different classes of IP service over the Internet, the IP differentiated services architecture defines three main

building blocks: packet classifiers, forwarding/per-hop behavior, and traffic conditioning policies. The differentiated services model utilizes static configuration of classification and forwarding policies in each node along a network path.

Services are built by applying rules – rules for how packets are marked initially and rules for how marked packets are treated at boundaries. At boundaries of domains, the only requirement is to have a bilateral agreement between the parties on each side of the boundary (i.e. no multilateral agreements are required). Three elements work in concert to deliver a DiffServ service: per-hop behaviors (PHBs) deliver special treatment to packets at forwarding time, traffic conditioners alter packet aggregates to enforce rules for services, and bandwidth brokers (policy managers) apply and communicate policy.

DiffServ exploits the distinction between edge and core devices for scalability reasons. It calls for pushing major state information and work to the edges of the network, while forwarding is expected to be done extremely quickly in the core of the network. Customers conclude contracts with their ISPs for specific QoS profiles: Service Level Agreements (SLA). Policing is done at the periphery of the DiffServ network domains. Relatively simple, differentiated per-hop forwarding behaviors (PHB) are indicated in the IP packet headers and applied to PHB traffic aggregates. The PHBs and the policing define the range of services offered. Each ISP domain contracts for aggregate QoS traffic profiles with neighboring ISP domains. Policing is done at the domain boundaries. Hence, the DiffServ concept supports simple, bilateral business agreements.

A Service Level Agreement defines the forwarding service the customer will receive and includes a Traffic Conditioning Agreement (TCA). The TCA defines what the client must do to achieve the service desired and what the service provider will do to enforce limits. A client may even be another service provider. An SLA may or may not specify the scope of the agreement, i.e. destinations in the provider network and what happens if the traffic goes beyond the provider domain.

Each packet receives a particular forwarding treatment based on the marking in its IP TOS octet (now called DS codepoint). The packet may be marked anywhere in the network, but is most probably marked at domain boundaries. The packet is treated the same way as all the others marked the same. There is no per-flow state required inside the network because core devices know only markings, not flows.

Per-flow state is kept at the network edge: flows are aggregated on the basis of the behavior desired. Please note that this scheme requires overall network engineering to ensure that aggregates get the appropriate or desired services.

In the DiffServ framework, packets carry their own state in a few bits of the IP header (the DSCP), which also leads to scalability of this QoS mechanism, making it appropriate for end-to-end QoS. Policy decisions and implementations are left to local “trust” domains. Traffic from clients is classified according to the SLA, possibly shaped at the boundary of the domain, and labeled with a DSCP. The ingress DS node assures conformance with the TCA. Each node selects the per-hop forwarding behavior on the basis of the codepoint. An egress DS node may perform traffic conditioning to reflect a TCA with the next domain.

The DiffServ architecture offers a framework within which service providers can provide their clients with a range of network services that are differentiated on the basis of performance in addition to pricing of tiers as used in the past. These services are monitored for fairness and compliance with the service agreements. In order to deliver service agreements, each DiffServ-enabled edge router implements traffic conditioning function which performs metering, shaping, policing, and marking of packets to ensure that the traffic entering a differentiated services network conforms to the TCA.

Traffic conditioners enforce the rules of each service at the network node input with the following functions (cf. Figure 3.5):

- Classifiers: both DSCP (TOS octet for IPv4) and general packet header (deep classification).

- Policers: token bucket with various actions.
- Shapers: enables customer to make aggregate flows conform.
- Markers: set DS byte based on classification.

Traffic Classification

Figure 3.5 illustrates the traffic classification and conditioning process in the ingress DS nodes. Not all of the elements depicted in the figure have to be present in all boundary nodes. The classifier selects packets based on the combination of one or more predefined sets of header fields. The mapping of network traffic to the specific behaviors that result in different classes of service is indicated by the differentiated service (DS) field shown in Figure 3.6. Each value of the DS field uniquely identifies the per-hop behavior or the treatment given to the traffic at each hop along the network path. The DiffServ architecture supports a maximum of 64 classes of service. Each router sorts the packets into queues on the basis of the given DS field. The queues may be treated differently depending on their priority, share of bandwidth, and discard policies.

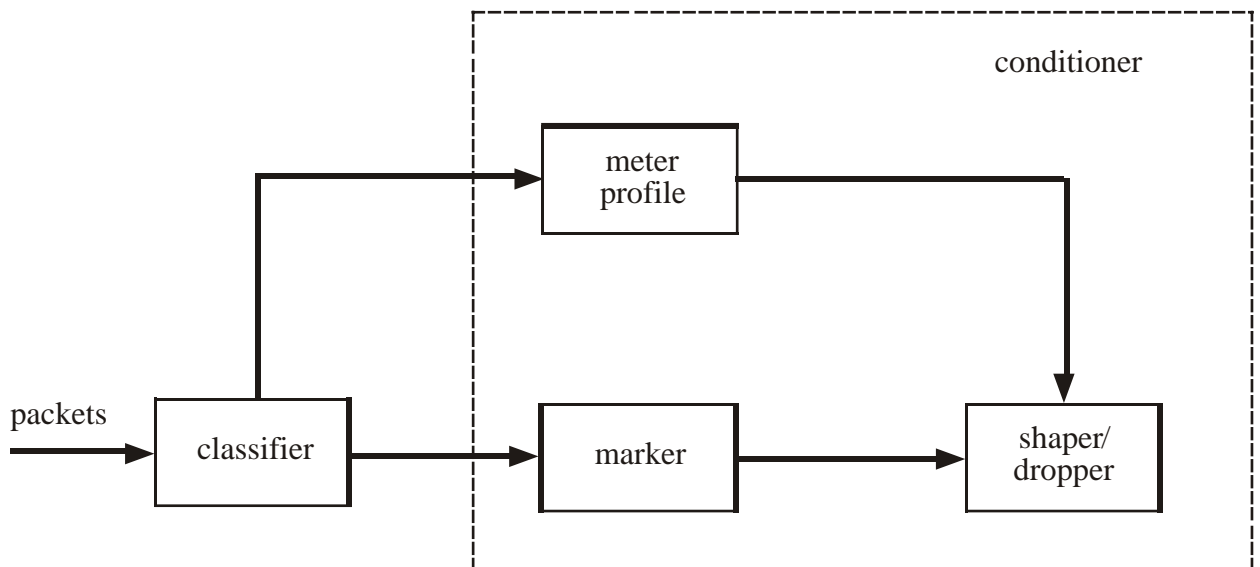


Figure 3.5: DiffServ Traffic Classification and Conditioning

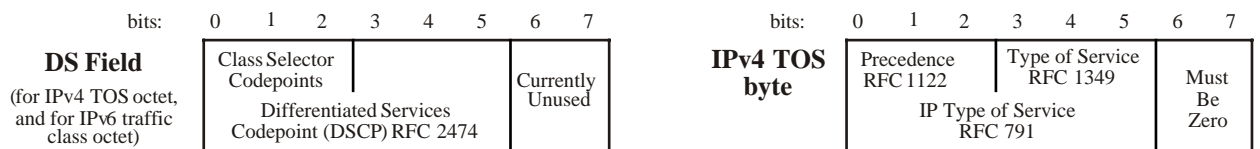


Figure 3.6: DiffServ Codepoints (DSCP)

The type of service (TOS) field – the DiffServ codepoint (DSCP) – is an 8-bit field, with the last two bits reserved. With the 6-bit DSCP, there are 64 possible codepoints: 48 in the global space, 16 for local use. Host vendors want to be able to set DSCPs and may also want to check and possibly reset DSCPs at domain boundaries.

The DS byte restructures the TOS field in the IPv4 header to permit use of parameters relevant for specified service levels and traffic behavior controls. Of the eight bits in the field, six bits define the per-hop behavior (PHB) the packet will receive with respect to the policies established at a network boundary and the other two bits are currently unused (CU). Figure 3.6 shows the Ipv4 datagram, including the DS-type field.

At the edge of the network or administrative boundary, the classifier determines the value of the DS field for each incoming flow.

On the basis of the DS field, these incoming flows are aggregated over an outgoing flow. The router implementations in the intermediate nodes use this 6-bit PHB field to index into a table for selecting a particular packet-handling mechanism. This forwarding policy determines how routers will handle the packets in terms of providing a class of service by combining traffic management functions such as packet queuing, scheduling, and buffer reservations at each node.

Traffic Conditioning

Traffic conditioners enforce the rules of each service at the network node input. The TCA agreement encompasses all of the traffic conditioning rules specified within a Service Level Agreement.

Metering: monitors the traffic pattern of each flow against the traffic profile. For out-of-profile traffic, the metering function interacts with other components to re-mark or drop the traffic for that flow.

Marking: customers request a specific performance level on a packet-by-packet basis by marking the DS field of each packet with a specific value. This value specifies the per-hop behavior (PHB) to be allocated to the packet within the provider's network. The edge routers classify the packets to identify the PHB and a DS codepoint for that packet.

Policing: at the ingress edge routers, the incoming traffic is classified into aggregates. These aggregates are policed as per the TCA. The out-of-profile traffic is either dropped at the edge or is re-marked with a different PHB.

Shaping: the routers control the forwarding rate of packets so that flow does not exceed the traffic rate specified by its profile. The shapers ensure fairness between flows that map to the same class of service and control the traffic flow to avoid congestion.

Per-Hop Behavior Policies (Forwarding)

PHBs are basically the packet forwarding treatment that delivers the differentiated service to packets at the network node output: policing, shaping, possible re-marking of DS codepoints, enqueueing treatment (e.g. drop preference), scheduling. PHB is the description of an externally observable forwarding behavior at a node and may be in terms of delay, jitter or drop priorities. A PHB group is a set of PHBs that work together based on a shared constraint such as sharing a queue or resource, e.g. high-priority and low-priority queues. When a node supports more than one PHB, the behavior relative to other PHBs has to be described.

Currently the IETF is defining the following PHBs: expedited forwarding (EF), assured forwarding (AF), and default (DE).

Expedited forwarding requires that (at each node) the egress rate exceed the ingress rate for a conforming aggregate. This is like a "virtual leased line". The EF treatment polices and drops on network ingress and shapes on egress to maintain the service contract to the next provider. Modest buffering is needed (no burst) and some form of priority queuing required. This treatment should be kept to a small fraction of total network traffic. This is a proposed IETF standard.

Assured forwarding defines four priorities of traffic receiving different bandwidth levels (the "Olympic services": Gold – Silver – Bronze – Best Effort). There are three drop preferences each (similar to Frame Relay in this respect). The lower the drop preference assigned the traffic, the higher the chance of it being dropped during congestion. There are token bucket policers for each priority (burst allowed). Enqueueing uses RED or a similar mechanism to distinguish drop preference and control congestion and scheduling based on the bandwidth of the priority.

The IETF DiffServ group has defined two service classes for supporting applications. The Premium Service model emulates the conventional leased-line service that promises to deliver customer traffic with a low loss probability at a given peak rate and a strict priority-based per-hop behavior for ensuring low latency. This service is suitable for applications that have strict bandwidth, latency, and jitter requirements. To create a low-loss, low-delay service, nodes must be configured so that the aggregate has a well-defined minimum departure rate, independent of other traffic. The second service model, Assured Service, emulates a lightly loaded network even in the presence of congestion. This service promises to deliver traffic with a high degree of reliability within the negotiated latency limits. The assured forwarding PHB group provides delivery of IP packets in four independently forwarded classes with three discard-precedence values for each class.

DiffServ expects advance provisioning and reservations to be made in each of the intermediate nodes along the network path. If a network path crosses multiple DS domains or multiple ISPs, the ISPs must support the same PHBs to provide consistent end-to-end service.

3.1.4.4 QoS Policies

In an open and public Internet (as well as in large Intranets), the acceptance of QoS requests results in better network service to some flows – possibly at the expense of service to traditional best-effort flows. This may be implemented by preferential queuing or dropping, admitting or denying access, or encrypting the packet's payload, to cite just a few examples. Protocols that explicitly support some or all of these functions include COPS, RADIUS, RSVP, IntServ, DiffServ, ISSLL, DSSLL, and IPSec. The successful wide-scale deployment of these and other protocols depends on the ability of the administrator of a network domain to administer and distribute consistent policy information to those multiple devices in the network which perform the classification and packet conditioning or treatment. Protocols that could be used for the distribution of the policy include LDAP, COPS, SNMP, and TELNET/CLI. The multiple types of devices that need to work in concert across even a single domain to achieve the desired policy can include hosts (clients and servers), routers, switches, firewalls, bandwidth brokers, subnet bandwidth managers, network access servers, and policy servers.

The IETF Policy working group is tackling how to represent, manage, and share policies and policy information in a vendor-independent, interoperable, scalable manner for QoS traffic management. See the working group's page at <http://www.ietf.org/html.charters/policy-charter.html>.

Mechanisms

Policy is comprised of the following three functions:

Decision-making: this compares the current state of the network to a desired state described by an application-specific policy and decides how to achieve the desired state.

Enforcement: this implements a desired policy state through a set of management commands. When applied to network elements, the management commands change the configuration of the device, enforcing QoS using one or more mechanisms. These mechanisms may be vendor specific.

Policing: this is a continual active or passive examination of the network and its constituent devices for checking network health, whether policies are being satisfied, and whether clients are taking unfair advantage of network services.

Decision-making uses static and/or dynamic data to determine if a type of policy is being satisfied and, if not, which steps are needed to satisfy that policy. Enforcement refers to the interpretation and execution of policies by consumers who are assumed to be trustworthy. Policing is the auditing of policy compliance to verify that the policy consumers properly implement policies.

Example: Provide the JitterFreeMPEG2 video service for authorized users between authorized points, but only at agreed-upon times. The policy condition could be loosely translated as: IF the

user is a member of an approved group (ApprovedUsers that are authorized to have this service) AND the service requested is one supported (VideoServices group) AND the source of the request is approved (in the VideoSources group or has been authenticated) AND the destination is approved (in the VideoDestinations group or has been authenticated) AND the time requested is OK (in ApprovedTimePeriods).

So far, the following elements have been identified as key elements of a policy architecture:

Data Store (DEN, DTMF)

This repository may be, but is not limited to, a directory accessed using the LDAP protocol. The Directory Enabled Network (DEN) Initiative and related specification work is an effort to build intelligent networks and networked applications that can associate users and applications to services available from the network according to a consistent and rational set of policies. DEN defines a directory as a centralized repository that coordinates information storage and retrieval, enabling other data- and application-specific repositories to be united. Eventually, intelligent network applications will transparently leverage the appropriate information about the network and the services that it offers on behalf of its users and the particular context in which the application is running. For more details, see the DEN FAQ at the Desktop Management Forum site www.dmtf.org.

Policy Decision Points (PDP)

This is the point where policy decisions are made.

Policy Enforcement (PEP)

This is the point where the policy decisions are actually enforced. It is assumed that policy decisions will always be made in the PDP and implemented in the PEP. Specifically, the PEP is not able to make decisions on its own, which simplifies the definition and modeling of policy.

Policy Protocols (COPS)

The Common Open Policy Service (COPS) protocol is emerging as a viable solution for distributed policy management. Initially, COPS will be used within a domain for router policy enforcement points to retrieve policy from the policy distribution points. COPS may also be used between bandwidth brokers (BBs) – which essentially act as PDPs – for dynamic inter-domain policy exchange. Bandwidth brokers may be third parties that manage SLAs for various ISPs and enterprises. Border exchanges of data between administrative domains can be policed, shaped, and conditioned according to SLAs that are encoded in a generic policy grammar.

Bandwidth Brokers (BB)

Bandwidth brokers for handling QoS policies are being defined in association with DiffServ. They are designed to be configured with organizational policies, keep track of the current allocation of marked traffic, and interpret new requests to mark traffic in light of the policies and current allocation. They are intended to be used to allocate bandwidth for end-to-end connections with fewer state and simpler trust relationships than deploying per-flow or per-filter guarantees in all the network elements on an end-to-end path.

Organizationally, the BB architecture is motivated by the observation that multilateral agreements rarely work. And this architecture allows end-to-end services to be constructed out of purely bilateral agreements. BBs only need to establish relationships of limited trust with their peers in adjacent DiffServ domains, unlike schemes that require the setting of flow specifications in routers throughout an end-to-end path.

BBs have two responsibilities: primarily, to parcel out their region's marked traffic allocations and set up the leaf routers within the local domain and, secondarily, to manage the messages that are

sent across boundaries to BBs in adjacent regions. A BB is associated with a particular trust region, one per domain. A BB has a policy database that stores the information on who can do what and when. BBs use their databases to authenticate requesters. Only a BB can configure the leaf routers to deliver a particular service to flows, which is crucial if a secure system is to be deployed. An initial request might cause communication between BBs in several domains along a path, but each communication is effected solely between two adjacent BBs. Initially, any agreements will be prenegotiated and fairly static. Some may become more dynamic as the service evolves.

3.1.4.5 IETF End-to-End QoS Model

In conclusion, Figure 3.7 shows how the different pieces of the work within the IETF might fit together in future networks. The integrated service model should be used at the edges of a network and the differentiated services model at the core of a network.

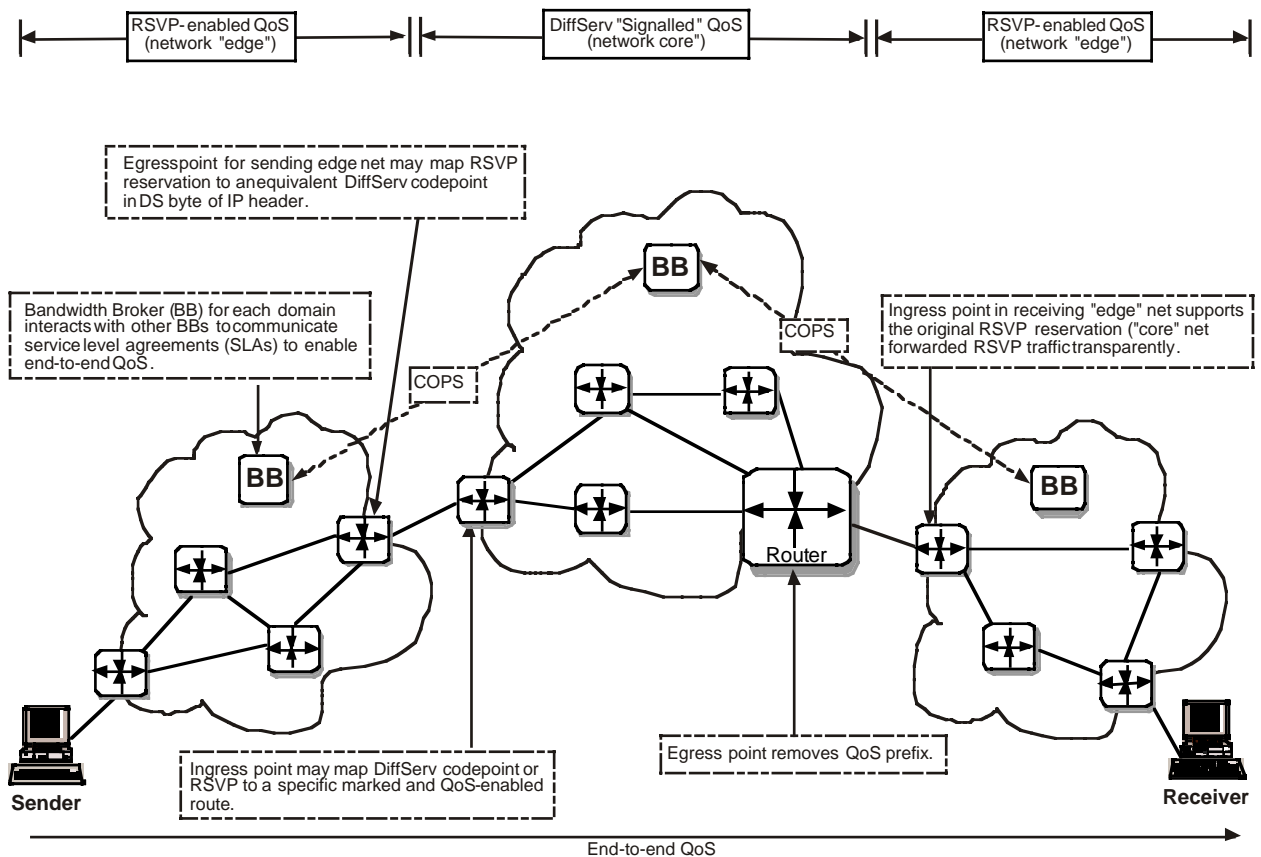


Figure 3.7: IETF End-to-End QoS Model

3.1.5 Microsoft Windows Socket API Version 2

The Winsock 2.2 API [WSG97a] [WSG96] differentiates two interfaces: the application programming interface (API) and the service provider interface (SPI) [WSG97b]. The API provides the user with transparent access to layer 3 and 4 communication services. The protocols themselves are implemented within the service providers. This includes transport service providers (TSP) and name space service providers (NSSP). Figure 3.8 gives an overview of the Winsock 2 architecture as the key component of the Windows Open System Architecture (WOSA).

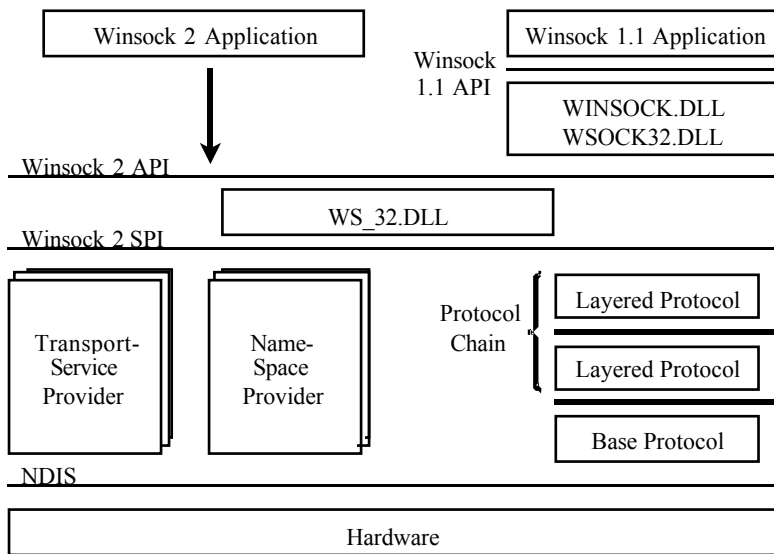


Figure 3.8: Winsock 2 as the Central Element of WOSA

Winsock 2 includes mechanisms enabling applications to negotiate quality of service with the network, thus facilitating multimedia applications for the user. Such applications can discover and utilize the quality of service (bandwidth, latency, etc.) offered by underlying networks such as ATM, RSVP or ISDN. Winsock 2 also provides generic support for both multipoint and multicast communication. This enables applications to discover and utilize capabilities such as IP-multicast and ATM-point-to-multipoint communication independent of the protocol. Applications can discover what type of multipoint or multicast capabilities a transport provides and use these facilities generically.

Winsock 2 extends the TCP/IP-centric socket paradigm to a generic interface away from protocol-specific functionality. Hence, a number of new functions have been added, all of which are designated by the prefix “WSA”.

Winsock 2 offers the application programmer two ways of specifying QoS: a protocol-specific QoS specification (e.g. compatible with UNI 3.x in the case of ATM) and, as an alternative, the Winsock 2 QoS flowspec. The flowspec is in accordance with the flow specification developed by Partridge [Par92].

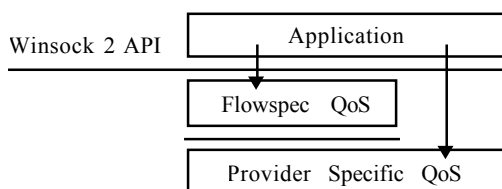


Figure 3.9: Protocol-independent Winsock 2 QoS (“Flowspec”)

The abstraction from protocol-specific QoS definitions provides the application programmer with a uniform QoS interface. The mapping of the flowspec to the protocol-dependent QoS parameters (e.g. traffic descriptor of ATM) is performed within the protocol-specific SPI.

The flowspec of the Winsock 2 API is described in more detail in appendix 11.5.

3.1.6 RSVP-aware Communication APIs

The most prominent RSVP-aware APIs are the RSVP API (RAPI) and the Winsock 2.2 API. These two APIs provide an API which enables multimedia applications to take advantage of the RSVP protocol.

RSVP API (RAPI)

RAPI [BrHo97] is a realization of the generic API contained in the RSVP functional specification [BZBHJ97]. RAPI is based on a client library linked with the application with RSVP implemented on a host by a user-level daemon program. The procedures of the RSVP client library module interact with the local RSVP daemon program through a UNIX-domain socket. RAPI describes the interface between the application and the RSVP client library.

In RAPI implementations, applications use a series of function calls to establish and maintain an RSVP session. To establish an RSVP session, the application uses the `rapi_session()` function. On the data sender side, the application calls `rapi_sender()` and, on the receiver side, the `rapi_reserve()` function to make a QoS reservation. `rapi_release()` is used for the closing of the session.

There are a number of RAPI implementations available [SuSo97].

Winsock 2 and RSVP

The RSVP annex to the Winsock 2 API specification describes an API that is specific to RSVP and is designed for applications wishing to exploit full RSVP functionality not covered by the generic QoS interface provided by Winsock 2. The RSVP annex can be used to invoke QoS on behalf of another application's data flows.

The event-driven programming approach of Winsock 2 is especially useful for supporting the asynchronous signaling of RSVP. Hence, only nonblocking `WSASockets` are used with RSVP. If a host wants to participate in an RSVP session, the host uses `RSVP_REGISTER`. With this call, the host states whether it wants to be a sender, receiver, or both in the multicast group. The RSVP messages `PATH` and `RESV` are controlled through the `RSVP_SENDER` and `RSVP_RESERVE` calls. The QoS specification (TSPEC or FLOWSPEC) is included in the calls. If an RSVP event occurs, the application is informed and receives an event description via `RSVP_RETRIEVE`. Policy control is planned but not yet implemented.

Intel PC-RSVP

PC-RSVP [IAL97] is a Winsock 2-layered service provider. It enhances a standard TCP/IP or UDP/IP service provider to a QoS-enabled service provider. In contrast to the RSVP annex of the Winsock 2 specification described above, PC-RSVP uses the generic QoS interface of Winsock 2. PC-RSVP provides applications with a transparent interface to conduct RSVP-related operations only on their behalf. Through the generic QoS interface, PC-RSVP supports limited RSVP functionality in comparison with the RSVP annex.

3.2 QoS Architectures

As pointed out in a survey of QoS architectures [AuCaHa98], a considerable amount of research has been done within the field of QoS support for distributed multimedia systems. The survey gives a comprehensive overview of the state of the art in the development of QoS architectures. In contrast to the rather isolated work which has been done within the context of individual architectural layers such as the distributed system platform, operating system, and communication system, QoS architectures are trying to set up a consistent framework to incorporate QoS

mechanisms across all the architectural layers and end-to-end. As described in the following sections, QoS-A, OMEGA, and HeITS have been the most prominent research efforts with an overall architectural approach in mind.

3.2.1 QoS-A

A layered architecture of services and mechanisms for quality of service management and control of continuous media flows in QoS-capable communication networks, the Quality of Service Architecture (QoS-A) [Camp96] incorporates three main concepts: (1) *Flows*, which characterize the production, transmission, and eventual consumption of single media streams (both unicast and multicast) with associated QoS. (2) *Service contracts*, which are binding agreements of QoS levels between users and providers. (3) And *flow management*, which provides for the monitoring and maintenance of the QoS levels contracted, where realizing the flow concept necessitates active QoS management and tight integration between device management, end-system thread scheduling, communications protocols, and networks.

In functional terms, the QoS-A (cf. Figure 3.10) comprises various layers and planes. Starting from the top down, the upper layer consists of a distributed applications platform augmented with services geared for the provision of multimedia communications and QoS specification in an object-based environment [CBDW92]. Below this lies an orchestration layer ensuring jitter correction and multimedia synchronization services across multiple related application flows [CCGH92]. Finally, basic support is given by a transport layer which contains a range of QoS-configurable services and mechanisms [CoCaRo95] and an underlying internetworking layer and lower layers which are to guarantee end-to-end QoS support.

In the QoS-A, quality-of-service management is realized in three vertical planes:

- The *protocol plane*, which consists of distinct user and control sub-planes, is motivated by the principle of separation. QoS-A uses separate protocol profiles for the control and media components of flows because of the different QoS requirements set for control and data.
- The *QoS maintenance plane*, which contains a number of layer-specific QoS managers which are to ensure the fine-grained monitoring and maintenance of the related protocol entities. At the orchestration layer [CCGH92], for example, the QoS manager is interested in the tightness of synchronization between multiple related flows. Yet, the transport QoS manager is solely concerned with intra-flow QoS such as bandwidth, loss, jitter, and delay. In fact, QoS managers maintain the level of QoS in the managed flow by means of fine-grained resource tuning strategies based on flow-monitoring information and a user-supplied service contract.
- The *flow management plane* is responsible for flow establishment (including end-to-end admission control, QoS-based routing, and resource reservation), QoS mapping (which translates QoS representations between layers), and QoS scaling (QoS filtering and QoS adaptation for coarse-grained QoS maintenance control).

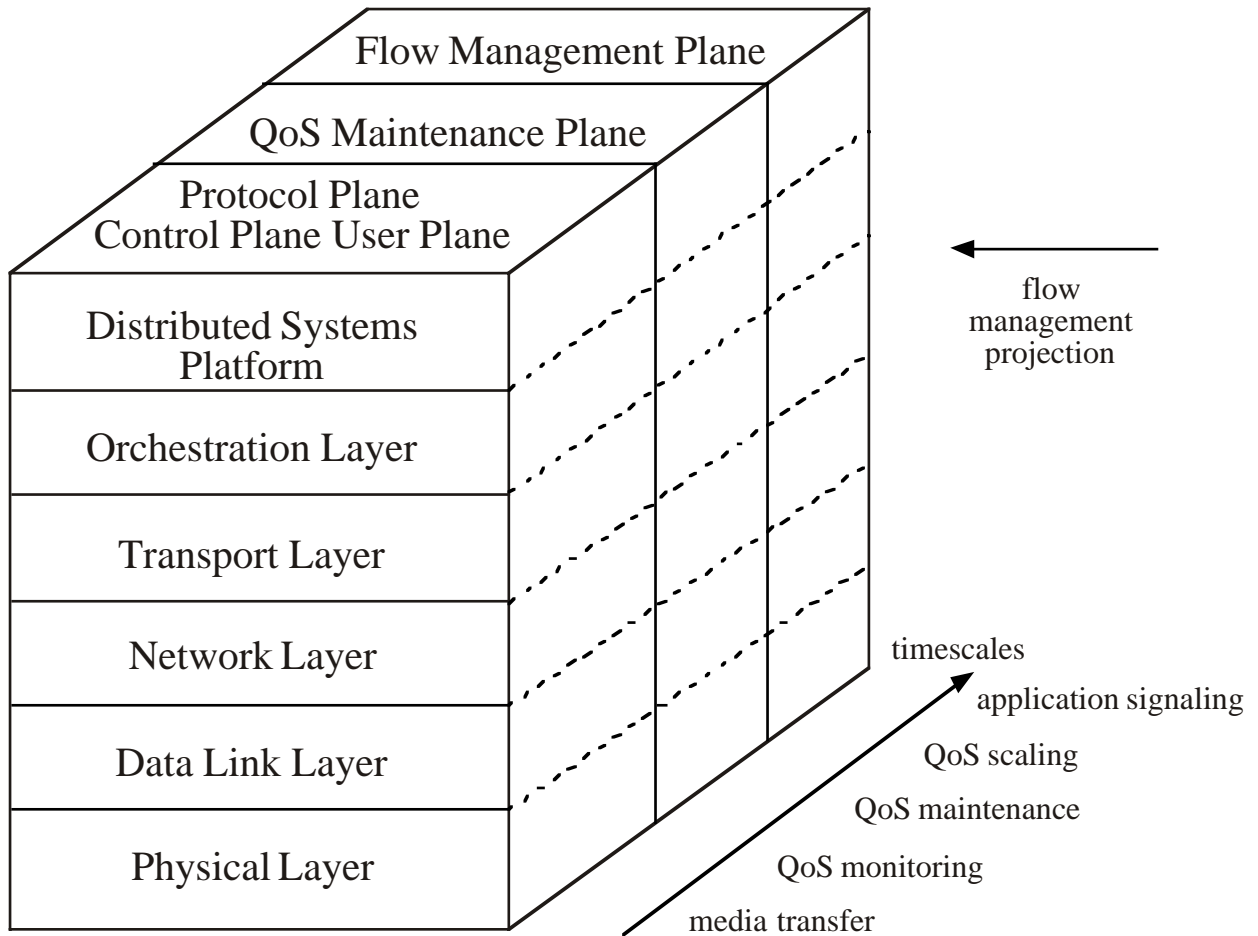


Figure 3.10: QoS-A

3.2.2 OMEGA

The OMEGA architecture [Na95], an endpoint architecture developed at the University of Philadelphia, is the result of an interdisciplinary research effort aimed at examining the relationship between application QoS requirements (which make stringent resource demands) and the ability of local (the operating system) and global resource management (combining communication and remotely managed resources) to satisfy these demands. As seen in Figure 3.11, the architecture is basically comprised of two elements: a network subsystem, which provides bounds on delay and errors and can meet bandwidth demands, and an operating system, which is capable of providing runtime QoS guarantees.

The essence of the OMEGA architecture is resource reservation and management of end-to-end resources. All communication is preceded by a call setup phase where application requirements expressed in terms of QoS parameters are negotiated. Guarantees are made at several logical levels, e.g. between applications and the network subsystem, between applications and the operating system, or between the network subsystem and the operating system. This leads to the establishment of tailor-made connections, thus resulting in the allocation of the resources needed to meet application requirements and operating system and/or network capabilities. To further facilitate this resource management process, the University of Pennsylvania has also implemented a *QoS brokerage model* [NaSm95] to incorporate QoS translation and QoS negotiation and renegotiation.

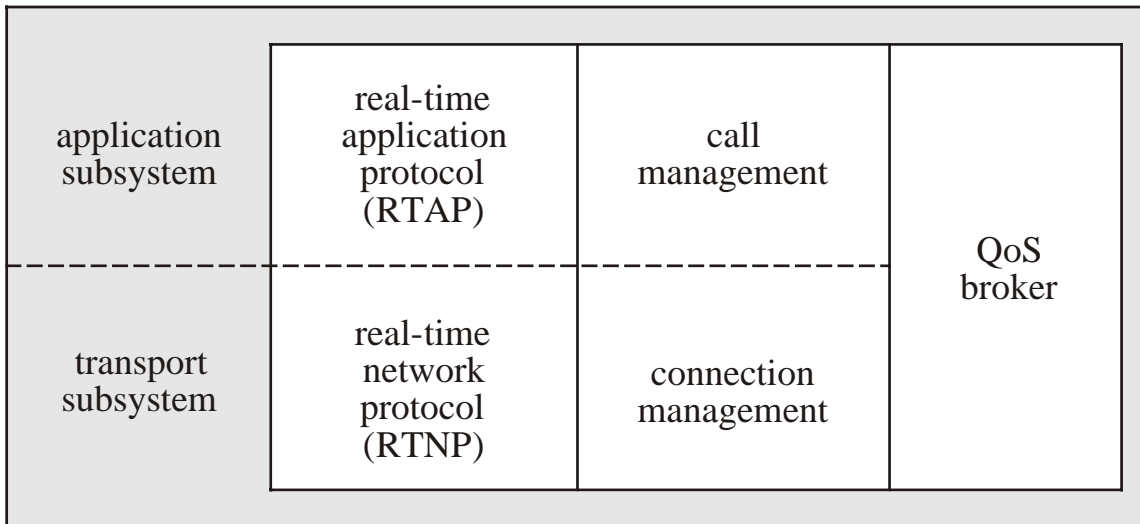


Figure 3.11: OMEGA QoS Broker

3.2.3 Heidelberg QoS Model

Industry is of course not neglecting work in this field: with the HeiProject, IBM in its European Networking Center in Heidelberg has developed a comprehensive QoS model which provides guarantees in the end systems and network [VWHW98]. As Figure 3.12 illustrates, the communications architecture includes a continuous media transport system (HeiTS/TP) [HHSS91] which provides QoS mapping and media scaling [DHH+93]. The underlying internetworking layer is based on ST-II [Top90] and supports not only guaranteed but also statistical levels of service. Also, QoS-based routing is supported via a QoS finder algorithm, and the network supports QoS filtering.

But the key to providing end-to-end guarantees is HeiRAT (Heidelberg resource administration technique) [VWHW98], which constitutes a comprehensive QoS management scheme including QoS negotiation, QoS calculation, admission control, QoS enforcement, and resource scheduling. The HeiRAT operating system scheduling policy is a rate-monotonic scheme with the priority of a system thread performing protocol processing proportional to the message rate requested.

Designed to handle heterogeneous QoS demands from individual receivers in a multicast group and to support QoS adaptivity via flow filtering and media scaling techniques, the Heidelberg QoS model is characterized by a basic principle: media scaling [DHH+93] and codec translation at the end systems, together with flow filtering and resource sharing in the network, are fundamental if heterogeneous QoS demands are to be met. Media scaling matches the source with the receivers' QoS capability by manipulating flows at the network edges, whereas filtering accommodates the receivers' QoS capability by manipulating flows at the core of the network as flows traverse bridges, switches, and routers. [Wolf96] contains an extensive discussion of the resource management scheme developed within the framework of this project.

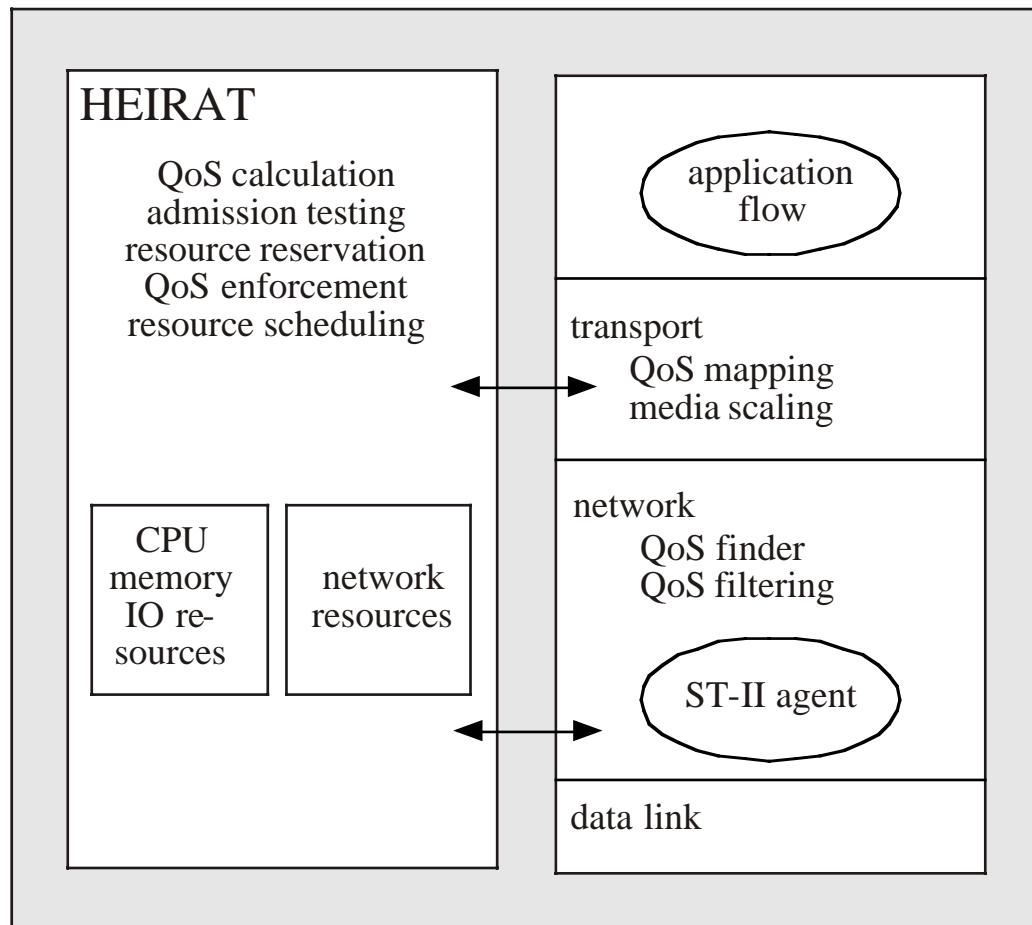


Figure 3.12: Heidelberg QoS Model

3.3 Related Work (Stream Control and Management)

Research recognizes the importance of controlling media flows and mapping quality-of-service parameters. In addition to proprietary solutions, work currently done seems to be focused in two directions: solutions which are based on the protocol family worked out in the IETF task force MMUSIC (Multiparty Multimedia Session Control) [IETF98] [Sch97] and solutions which are founded on the OMG A/V Streaming Specification [MGRO97].

The protocols specified in the MMUSIC task force are in various stages of development. In connection with the work being done, RTSP (real-time streaming protocol) [ScRaLa98] seems to be the most significant. RTSP uses the existing Web infrastructure and may be viewed as a protocol for the remote control of Internet VCRs. RTSP was developed in a cooperative effort by Real Networks, Netscape Communications, and Columbia University and has been proposed to the IETF.

Of the work done which uses the OMG A/V Streaming Specification as its foundation, the A/V extensions introduced by TAO [MuSuSc99] and Ionas OrbixMX [IT98] and the work done by NEC C&C [WSRR98] are most noteworthy. These prototypical implementations are all partial implementations of the OMG A/V Streaming Specification and are in too early a stage of development to allow a decision to be made as to how they might be integrated as components into the overall system.

3.3.1 ORB-based Approaches

As already introduced in section 2.3.1, the Common Object Request Broker Architecture (CORBA) [OMG95] defined by the Object Management Group (OMG) provides a standardized method of building distributed applications. OMG's Interface Definition Language (IDL) [ISO98] is used in a number of proposals to define interfaces between the parts of distributed applications. This section introduces the OMG A/V Streaming Specification and two related activities which are relevant for this work.

3.3.1.1 OMG A/V RFP

The OMG A/V Streaming approach stems from the activities developed in the Telecommunication Information Networking Architecture Consortium (TINA-C). This approach separates audio and video streams from the control and management data flow which is carried out via CORBA [MGRO97].

OMG's Request for Proposal (RFP) for the Control and Management of Audio/Video Streams (telecom/96-09-01) contained 12 issues: topologies for streams, multiple flows, stream description and typing, stream interface identification and reference, stream set-up and release, stream modification, stream termination, multiple protocols, quality of service, flow synchronization, interoperability, security.

The response to OMG's RFP as submitted by Iona, Lucent, and Siemens-Nixdorf [MGRO97] addressed issues 1-11 and provided hooks for solutions to issue 12. The proposal was based on work done in the RACE II project EuroBridge [KiMu96].

The example stream architecture from the OMG "Control of A/V Streams" RFP is depicted in Figure 3.13. The architecture in the RFP submission is based upon this figure.

It shows a stream with a single flow between two stream endpoints, with one acting as the source of the data and the other the sink. Each stream endpoint, shown as a dotted encapsulation in the figure, consists of three logical entities: a stream interface control object which provides IDL-defined interfaces (as server, 2b) for controlling and managing the stream (as well as potentially, outside the scope of the submission, invoking operations as client, 2a, on other server objects), a flow data source or sink object (at least one per stream endpoint) which is the final destination of the data flow, (3) and a stream adaptor which transmits and receives frames over a network.

The stream interface control object is depicted in Figure 3.13 as using the basic object adaptor (BOA) that transmits and receives control messages in a CORBA-compliant fashion. The RFP submission does not require any changes to the BOA or IDL language bindings to accommodate the control of A/V streams.

When a stream is terminated in hardware, the source/sink object and the stream adaptor may not be visible as distinct entities. How the stream interface control object communicates with the source/sink object – and perhaps indirectly with the stream adaptor (interface 4) – and how the source/sink object communicates with the stream adaptor (interface 3) are outside the scope of the RFP submission.

The "Control of A/V Streams" RFP submission provides definitions of the components making up a stream and interface definition of stream control and management objects (interface number 1a) as well as interface definitions of the stream interface control objects (interface number 2b) associated with individual stream endpoints. In particular, CORBA interface references for stream interface control objects are used to point to all of the stream endpoints in the parameters of the stream control operations defined in the submission. Thus, the submission does not need to define a new IDL data type for stream interface reference.

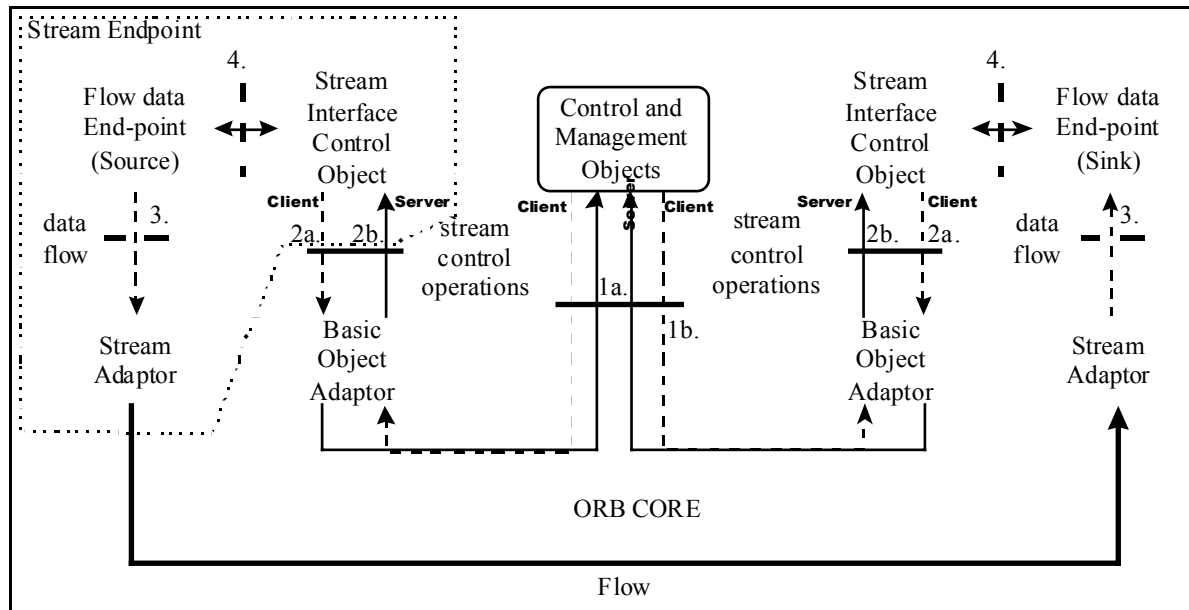


Figure 3.13: Example RFP Stream Architecture

Figure 3.14 illustrates some aspects of the OMG A/V stream architecture for a bi-directional video connection. A stream encapsulates several logical data streams of a kind in a single flow. The control of these objects is enabled by the interfaces “StreamEndPoint” and “FlowEndPoint”. Start, stop, connect, etc. are typical services that are provided by the interfaces.

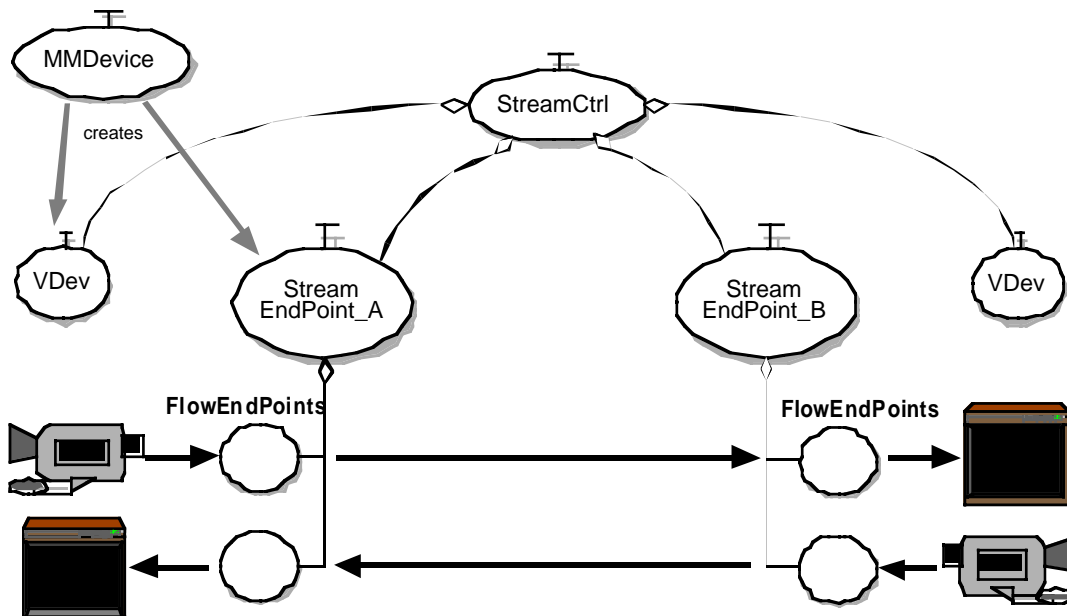


Figure 3.14: Example Bi-Directional Video Connection as per the OMG A/V Streaming Specification

The object MMDevice represents an abstraction of a multimedia device and owns an interface for the configuration of various encoding parameters. On the other hand, the interface permits a device to be bound to another device through one or more streams. The MMDevice returns a StreamEndPoint and a virtual MMDevice in order to support several connections. Stream-oriented control is enabled through the VDev interface.

The StreamCtrl object abstracts the continuous media transmission between virtual devices. In particular, it enables the binding between virtual devices, control commands (start, stop, pause, etc.), and configuration of QoS parameters for the communication system. Also, streams between stream endpoints without virtual devices are supported, thus enabling the media transfer between distributed application components.

3.3.1.2 TAO

Developed by Washington University [ScLeMu97], [SGHP97], the ACE ORB (TAO) is a CORBA-compliant ORB end system targeted for applications with deterministic and statistical quality-of-service requirements, and best-effort requirements. As the name shows, TAO was designed and realized using the ADAPTIVE Communication Environment (ACE) [Sch93], also developed at Washington University.

TAO focuses mainly on the topics related to real-time ORB end systems:

- Identifying the enhancements required for standard ORB specifications such as OMG CORBA and DCOM, thus enabling applications to specify their QoS requirements to ORBs.
- Empirically determining the features required to build real-time ORBs able to enforce the deterministic and statistical end-to-end QoS guarantees required by applications.
- Integrating the strategies for I/O subsystem architectures and optimizations with ORB middleware to provide end-to-end bandwidth, latency, and reliability guarantees to distributed applications.
- Capturing and documenting the key design patterns required to develop, maintain, and extend real-time ORB middleware.

Both TAO and its predecessor ACE have found their main field of application in systems which have a need for hard real-time guarantees, e.g. avionic systems. Some recent work illustrating how TAO may be deployed in A/V streaming may be found in [MuSuSc99].

3.3.1.3 NEC C&C Research Laboratories

NEC CCRL has reported on work using CORBA to provide virtual services and virtual interfaces to facilitate the development of distributed multimedia applications [WSRR98]. One example of such a distributed multimedia application is Cockpit View [OMRW97], which uses adaptive QoS for dynamic renegotiation of contract terms at each layer of the system. Virtual services are provided as high-level communication services, e.g. for multimedia stream handling. In most cases, virtual services contain a capability for guaranteeing QoS. Virtual interfaces provide an abstracted view of network resources such as ATM switches, encapsulating implementation details such as vendor-specific message formats and proprietary control protocols. The interfaces can include both control and management functions, thus integrating policies and network status information from the management side with QoS-sensitive control of management sessions.

3.3.1.4 Extended Integrated Reference Model (XRM)

The work being done by the COMET group at Columbia University comprises the development of an Extended Integrated Reference Model (XRM) [LaBhLi95] as a modeling framework for the control and management of multimedia telecommunications networks (which comprise multimedia computing platforms and broadband networks). In particular, this group proposes that the control and management, i.e. the operability, of multimedia computing and networking devices are equivalent. In fact, they argue that both classes of devices may be modeled as producers, consumers, and processors of media and see the sole difference as lying in the overall goal which a group of devices has to achieve in the network or end system.

As set out in Figure 3.15, the XRM is divided into five distinct planes [CHLL97], [Lazar92]:

- The *management function*, which resides in the so-called N-plane, the network management plane, and covers the OSI functional areas of network and system management.
- The *traffic control function*, which is made up of the resource control (M-plane) and connection management and control (C-plane) planes, whereby resource control takes on the tasks of cell scheduling, call admission, call routing in the network, process scheduling, memory management, routing, admission control, and flow control in the end systems.
- The *information transport function* located in the user transport plane (U-plane), which models the media protocols and entities for the transport of user information not only in the network but also in the end systems.
- And the *telebase*, which, in the data abstraction and management plane (D-plane), collectively represents the information and data abstractions found in the network and end systems. In particular, the telebase implements data sharing among the other XRM planes.

Based on the theory of guaranteeing QoS requirements in ATM networks and end systems populated with multimedia devices, XRM contains general concepts for characterizing the capacity of network and end-system devices (e.g. disks, switches, etc.). For example, at the network layer, XRM characterizes the capacity region of an ATM multiplexer with QoS guarantees as a *schedulable region*. Furthermore, network resources such as switching bandwidth and link capacity are allocated on the basis of four cell-level traffic classes (I, II, III, and C) for circuit emulation, voice and video, data, and network management respectively, with a traffic class typically characterized by its statistical properties and QoS requirements. QoS requirements, as a rule, reflect cell loss and delay constraints, so that, in order to best satisfy the QoS requirements of the cell level, scheduling and buffer management algorithms are needed to dynamically allocate communication bandwidth and buffer space accordingly.

A further characteristic of XRM is that flow requirements in the end system are modeled through service class specifications with QoS constraints. So, in the audio/video unit, the service class specification is given in terms of JPEG, MPEG-1, MPEG-2 video, and CD audio quality flows with QoS guarantees, and quality of service for these classes is specified by a set of frame delay and loss constraints. In fact, the methodology of characterizing network resources is extended to the end system to represent the capacity of multimedia devices. If the idea of a *multimedia capacity region* [HILY96] is applied, the problem of scheduling flows in the end system becomes identical to the real-time packing exercise of the network layer. XRM, together with its key resource abstractions (i.e. schedulable and multimedia capacity region), is currently being implemented as a component of a *binding architecture* [LaBhLi95] for open signaling, control, and management of multimedia networks.

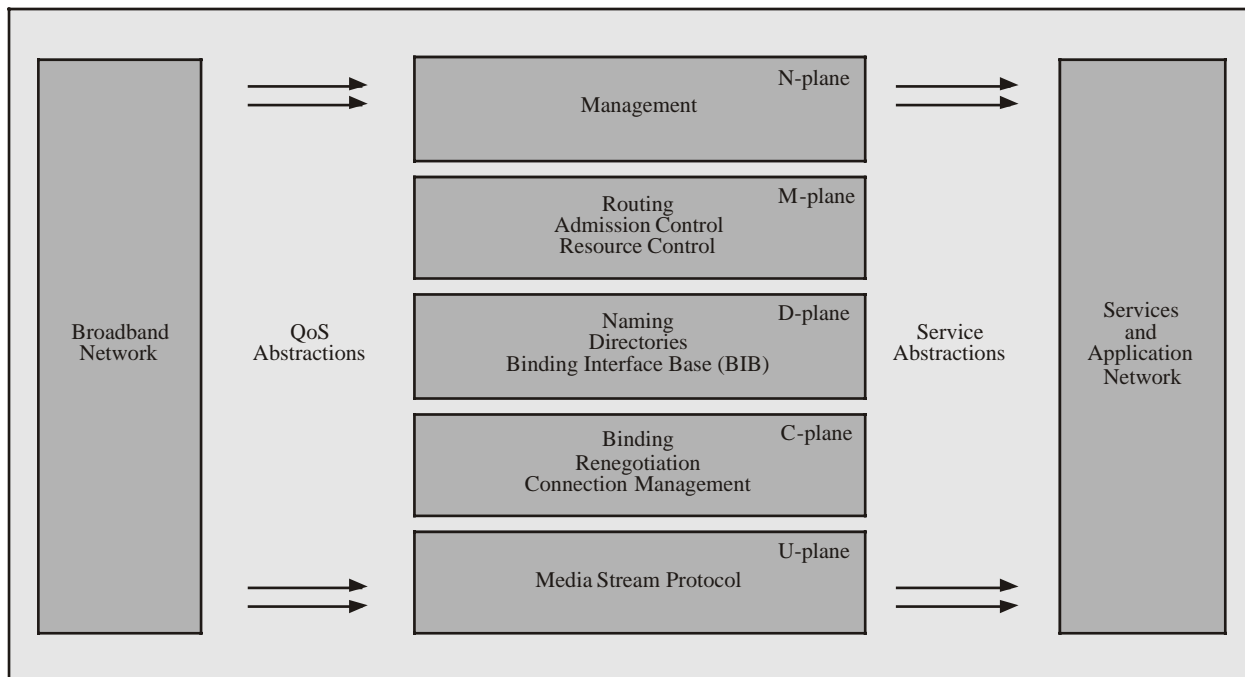


Figure 3.15: Extended Integrated Reference Model (XRM)

3.3.1.5 TINA

The TINA QoS Framework [TINA95] describes a framework for specifying QoS aspects of distributed telecommunications within the context of the Computing Architecture. The QoS framework addresses the computational and engineering viewpoints of distributed telecommunications applications. It is governed by the separation between telecommunication applications and the Distributed Processing Environment (DPE), i.e. multimedia services offered by a provider utilize the DPE and underlying computing and communications capabilities. The TINA QoS framework is partly based on work in the literature (e.g. ANSA QoS Framework [Guan94] and CNET Framework [HaHoSt93]). From the computational perspective, the QoS parameters required to provide guarantees to objects are stated declaratively as service attributes. In the engineering model, QoS mechanisms employed by resource managers are considered. By stating QoS requirements declaratively, applications are relieved of the burden of coping with the complex resource management mechanisms needed for ensuring QoS guarantees.

3.3.1.6 Open Signaling and Architecture Interest Group

The Open Signaling and Architecture Interest Group (Opensig/Openarch) has evolved out of the activities of the COMET group of Columbia University described in section 3.3.1.4. The charter of the working group targets research in understanding open network control issues as they arise in signaling, middleware, and service creation on ATM, Internet, and mobile multimedia networking platforms. The work of the Opensig group has focused on the definition, implementation, and experimentation of Open Programmable Networks. To have an impact on the products and services of next generation networks is a key objective of the Opensig working group. A vital step in this quest was the submission of a Project Authorization Request to IEEE by Columbia University, Ericsson, ISS, and NEC with the goal of standardizing “Programming Interfaces for Networks” (PIN). This request was approved on December 8, 1997. Information about this IEEE Communications Society standardization activity P1520 is available at the IEEE PIN web site [BHL+99].

3.3.2 IETF

Most of the activities in both the Internet and the Intranets have focused on guaranteeing quality of service (RSVP [BZBHHJ97]) and end-to-end transport using UDP or the Real-Time Transport

Protocol (RTP) [SCFJ96]. Each application uses its own application-level control protocol. In [Sch97], a proposal is put forth for a control architecture that not only offers the most widely used telephony features but also integrates multimedia storage, retrieval, and multimedia conferencing. The architecture includes two independent protocols: the Session Initiation Protocol (SIP) [HaScSc97] and the Real-Time Streaming Protocol (RTSP) [ScRaLa98].

RTSP is an application-level protocol for control over the delivery of data with real-time properties. In addition, RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Data is carried out-of-band by a different protocol, for example RTP.

Designed for the transport of real-time data, including audio and video, RTP consists of a data and a control part called RTCP. RTP uses two separate channels: one for the streaming data and the other for the coordination between client and server.

3.3.3 Development Environments for Multimedia Applications

3.3.3.1 Java Media Framework (JMF)

The Java Media Framework (JMF) is an API for incorporating time-based media into Java applications and applets. The JMF 1.0 API (the Java Media Player API) [Sun97] enabled programmers to develop Java programs that presented time-based media. The JMF 2.0 API [Sun99] extends the framework to provide support in capturing and storing media data, controlling the type of processing that is performed during playback, and performing custom processing on media data streams. In addition, JMF 2.0 defines a plug-in API that enables advanced developers and technology providers to more easily customize and extend JMF functionality. The high-level JMF architecture is depicted in Figure 3.16.

The classes in `javax.media.rtp`, `javax.media.rtp.event`, and `javax.media.rtp.rtcp` provide support for RTP, which enables the transmission and reception of real-time media streams across the network. RTP can be used for media-on-demand applications and for interactive services such as Internet telephony.

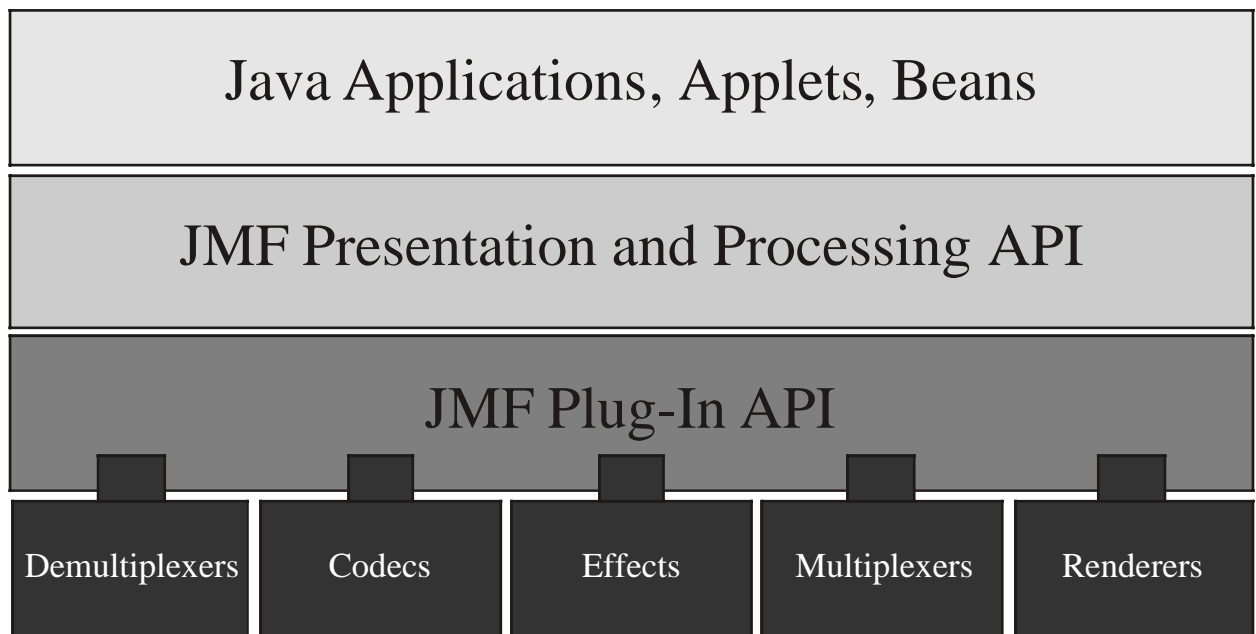


Figure 3.16: JMF Architecture

Within JMF, some player objects (JMF devices) which allow the playback of registered media types and the registration of custom media types and players are made available. Table 3.3 shows the media formats and protocols supported:

Audio	AIFF, AU, DVI, G.723, GSM, IMA4, MIDI, MPEG-1 Layer 1 &2, PCM, RMF, WAV
Video	Apple Graphics (SMC), Apple Animation (RLE), Cinepak, H.261, H.263, Indeo 3.2, Motion-JPEG, MPEG-1
Files	AVI, Quicktime, Vivo
Protocols	File, FTP, HTTP, RTP

Table 3.3: JMF 2.0 Media Formats [Sun99]

3.3.3.2 Configurable INtEgrated Multimedia Architecture (CINEMA)

The primary objective of CINEMA (Configurable INtEgrated Multimedia Architecture) was to develop a multimedia development platform which would enrich the given functionality by specialized abstractions and mechanisms which support distributed multimedia processing. The CINEMA development environment was built by the University of Stuttgart [BaHeRo95] [RoBaHe94]. The main issues that have been tackled include the communication of multimedia data under real-time conditions, the reservation of resources, and the synchronization of multimedia data streams. Providing such a development platform, CINEMA supports the development and control of multimedia applications with arbitrary processing topologies consisting of multiple data sources and sinks as well as arbitrary intermediate processing stages. Multimedia processing is done by processing components with generic interfaces. The CINEMA client merely implements the appropriate processing functionality without having to consider timing constraints or data transmission channels. Processing components may be nested to enhance the reusability of software and to establish higher functional abstractions. Typically, multimedia applications are created by interconnecting the data access points of processing components. Thus, a CINEMA client is enabled to build arbitrarily structured data flow graphs depending on the structure of the application. Before the flow of data units is started, the quality of service and synchronization relationships among data streams are specified, resulting in appropriate reservations of resources. So, a client defines the requirements and characteristics of the application to be built. At the same time, a client is supplied with abstractions allowing it to control the flow of data units and the structure of the application during run-time. Inside the CINEMA system, the requirements specified by clients are analyzed and a running system is set up transparently. The CINEMA system mainly covers three system services: configuration management, the synchronization service, and resource management.

3.3.3.3 Distributed Multimedia Object Services (DMOS)

In parallel to the more communication-centered work described in section 3.2.3, IBM's European Networking Center in Heidelberg has designed an approach for a development environment called DMOS [Käp97]. DMOS tries to facilitate the development of distributed multimedia applications through the introduction of a service interface which allows applications in a distributed system to access the data and processing components of multimedia data. This service interface is made available through IBM's DSOM, which is a very limited CORBA-compliant implementation with respect to available services. As depicted in Figure 3.17, the control plane is separated from the data plane. Also shown are the interprocess communication between the control and the data planes, which is performed by a proprietary protocol, and the communication between processes inside the data plane.

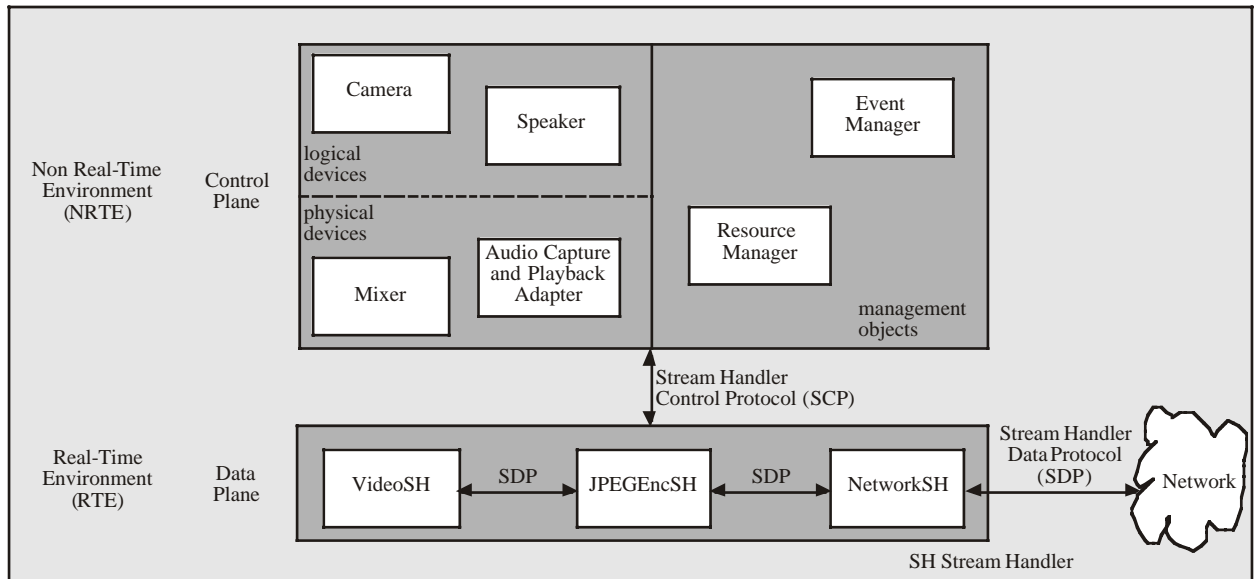


Figure 3.17: DMO Server Architecture

3.3.4 DSM-CC and DAVIC

DSM-CC (Digital Storage Media – Command and Control) is an ISO/IEC standard [ISO96] developed for the delivery of multimedia broadband services. DSM-CC defines an abstract basic reference model and the protocols needed to deliver a complete application such as video on demand or home shopping. Since DSM-CC is transport layer-independent, any application using DSM-CC does not need to be concerned with the underlying transport layer used between client and server. DSM-CC covers a number of distinct protocol areas: network session and resource control, configuration of a client, downloading to a client, VCR-like control of the video stream, generic interactive application services, and generic broadcast application services.

DAVIC, the Digital Audio-Visual Council, a non-profit organization with members from more than 200 corporations, defined one of the first uses of DSM-CC. DAVIC's mission is to promote the success of emerging digital audio-visual applications and services over various networks by the timely availability of internationally accepted interfaces and protocols. Hence, DAVIC has adopted DSM-CC in its DAVIC 1.0 specification [DAVIC95], where it is used as the protocol for control of multimedia interactive sessions and the resources within the sessions and for service-level interactions. The DAVIC 1.1, 1.2, and 1.3 specifications are in the process of extending the 1.0 specification to Internet access.

OMG's Interface Definition Language [ISO98] is used by DSM-CC to specify its interfaces. Although DSM-CC does not specify which Remote Procedure Call Scheme (RPC) is used, interoperability with CORBA is a goal. DAVIC has chosen CORBA 2.0 RPC (UNO – Universal Networked Objects) and encoding (CDR – Common Data Representation) for its specification of DSM-CC.

3.4 Active Networks and QoS

Research in active networks as recently evolved is an interesting new research topic [DFFR99]. The main idea behind the active network approach is to flexibly extend or change the functionality and behavior of end systems and intermediate network nodes through the injection of code in the network nodes. Traditional data packets are to be replaced by active packets – also called capsules – which can carry both user data and code. Intermediate nodes need to have a way to identify active packets and to load and execute the injected code in an appropriate execution environment.

According to [DFFR99], the benefits of active networks fall into the following categories: availability of information held by intermediate nodes, data processing capability along the path, adoption of distributed strategies, and easy development of new network services.

All of these potential benefits are extremely useful for building distributed multimedia applications and the provision of network QoS, as already shown in publications such as [BaPiRi98] [BETT97].

3.5 Summary

In this chapter the state of the art in de-jure and de-facto standards as well as related key research work dealing with QoS and stream control and management have been summarized and reviewed. As pointed out and summarized in [StWo97] [AuCaHa98] [SBLMP97], after several years of intensive research in QoS, a number of open issues still remain to be resolved:

- QoS support in end systems and servers for high-quality applications.
- Easy access to the QoS features of the communication and distributed system from an application programmer's point of view.
- QoS-driven routing algorithms.
- Pricing in multi-service and "QoS on demand" networks.
- Security mechanisms such as QoS support in firewalls, encryption or watermarking.
- Heterogeneity of communication systems, i.e. ATM vs. Internet protocols and QoS approaches.
- Scalability with respect to the number of participants in one application and scalability with respect to the number of concurrent applications.
- Resource reservation, including reservations in advance.
- Scaling, filtering, and application adaptation.
- QoS mapping.
- QoS support through active networks.
- QoS support in mobile systems.
- QoS support in distributed systems.
- End-to-end QoS framework and architecture.

It can be determined that end systems, servers, and their related operating systems are, in principal, ready to handle both low-quality and high-quality multimedia content. While high-quality multimedia content such as MPEG-2 necessitates additional hardware and software support, for low-quality multimedia content such as MPEG-1, H.263, Realmedia or Quicktime, software-only solutions are sufficiently performant.

The review of QoS-capable communication systems and the related research work done in section 3.1 indicates that QoS support of the communication system for desktop computers is still in its infancy. This is surprising because, to a large extent, QoS research in recent years has concentrated on communication systems. QoS-capable communication systems such as ATM are mature; but, in most cases, ATM has never reached the end system on the desktop. The reasons are manifold: its complexity in comparison with Ethernet, its high price per port at the beginning, interoperability

with existing network infrastructure, to name just a few. This has led to ATM being used mostly in backbone networks only. ATM has made its way to the desktop solely in restricted and local areas, e.g. video processing in production studios with a demand for high-volume real-time transfer of multimedia content. The Internet IntServ and the DiffServ architectures are still in early stages of development and deployment. The inherent scalability problems of both IntServ and IP multicasting remain to be solved. DiffServ is in a very early design stage. The interoperability between different QoS-capable communication systems such as IntServ, DiffServ, and ATM is still an open issue [SWKS99] [GuFaVe98].

My experience with the Winsock 2 API using SPIs for native ATM and RSVP and implementations of the RSVP API showed that the abstraction layer of the sockets is too low and complex for application programmers. Therefore, I propose the extension of the socket API used in Java with QoS support for RSVP and native ATM services [Ebe98b] [ERR98] as described in further detail in the following sections.

With respect to end-to-end integrated QoS architectures, generally agreed upon solutions are not available. The reason might be the overall complexity of the universe of discourse and the enormous variety of open issues as outlined above.

The related work on multimedia stream control and management has shown that there is at least a consensus that the multimedia stream should be separated from its control and management. The ORB-based approaches discussed in section 3.3.1 have shown exceptionally promising results, but they are still in their early stages of development. The IETF work on streaming media control as discussed in section 3.3.2 is well suited for simple low-quality video streaming applications. However the work is only geared for low-level transport protocols, i.e. it should be made available on a higher level of abstraction as proposed in this thesis and in JMF (cf. section 3.3.3.1). Unfortunately, JMF does not provide QoS to programmers of distributed multimedia applications. The CINEMA system basically covered only three system services: configuration management, synchronization service, and resource management. To build a distributed multimedia application, it uses its own proprietary instantiation and communication mechanisms. Similar to CINEMA, DMOS makes use of proprietary interprocess communication mechanisms, which makes them exceptionally inefficacious with respect to further advances in distributed systems.

In contrast, this work proposes the use of commercial off-the-shelf software such as a CORBA-compliant ORB, which is a proven technology for building distributed applications in heterogeneous environments. Unfortunately, the current CORBA specifications and other similar middleware technologies such as DCOM and RMI [OrHa97] only support applications with best-effort requirements. Existing middleware technologies and implementations are not suited for high-performant and distributed real-time applications such as multimedia applications. Interfaces with QoS specification, QoS enforcement, and throughput optimization are not standardized [SBLMP97]. Currently there is no agreed-on and consistent way of specifying and managing QoS in distributed systems [OMG97]. This thesis is contradictory to other work which proposes the extension of CORBA with QoS extensions [BeGe99], because it proves that it is sufficient to use CORBA exclusively for stream management and control and interprocess communication. Of course, QoS extension for middleware makes sense for systems with real-time requirements such as avionic systems, which do not have to transport high-volume data such as video at the same time as control information.

The separation of stream control and management from the transfer of the multimedia stream itself actually does not contradict the activities in active network research, where data and code are put together in capsules. As discussed in section 5.6, the active network approach meshes very well with the architecture proposed in this thesis.

4 Requirements Analysis

As described in chapter 1, the application background of this thesis is teletraining and distance learning. This chapter analyzes the requirements of these kinds of applications with respect to distributed multimedia applications. This application area will be used to validate the architecture described in the next chapter.

A huge percentage of companies still use traditional classroom-based training to school their employees. Until recently, only a few companies have used additional means of providing training to their employees such as business TV or CDs as delivery mechanisms. In contrast to these independent approaches, the training concept presented in this thesis integrates the following types of training courses seamlessly:

- Traditional classroom-based training with or without access to training material through the media server.
- Business TV-based teletraining with access to training material through an on-line media service.
- CD-based computer-based training (CBT) without any further on-line support or access to a training or media service.
- Web-based training (WBT) with full on-line access to training curricula and on-line assistance through the Intranet and Internet.

For reasons of effectiveness and efficiency, these different types of trainings are integrated in a curricular approach, which can last from one week up to two months. By using an integrated approach, the strengths and weaknesses of each training style can be exploited. For example, traditional classroom-based training is best for hands-on training measures, but needs a huge number of trainers and extensive training facilities. With business TV-based training, a large audience can be reached at the same time in a timely fashion, for example, to train sales personnel upon the introduction of a new car model. Of course, due to the usual one-way delivery mechanism of satellite systems, the lack of interactivity between trainer and trainees is a handicap. The advantage of CD-based training is, for example, that it does not require a sophisticated communication infrastructure as is the case with Web-based training. But WBT has the advantage that it can be updated very easily with new training material. Both CD- and Web-based training are suitable for self-paced training at the workplace or at home. These are only a few pros and cons of each training style - most important is to seamlessly combine them in an integrated didactic concept.

4.1 Learning Process

As illustrated in Figure 4.1, the overall process of on-line access to training content includes, at the least, the following steps, which are covered by a learning management system:

- Learning management: The people responsible for management of the system are authorized to create user groups, to add users to certain groups, to extend the system by adding a new functionality, to implement billing, and to take care of quality assurance.
- Content analysis and design: This includes the steps of analyzing the training needs (e.g. long-term training, introduction of a new product or of a new technology), course conception (selection of the appropriate course modules), course design (which types of training are to be applied in training modules), and media design.

- **Content production:** Within this phase, the on-line course parts are created, content is produced under consideration of the different media types and tests, and animations and simulations are created.
- **Content storage, retrieval, and delivery:** This covers the storage of all kinds of training material such as audio, video, text, animations, and simulations grouped in fine- or coarse-grain units of learning material.
- **Content reception by the learners:** The learning content is delivered to the user via different types of training. Embedded in the training, the WBT type not only provides the structured on-line course content but also offers communication means, tools for investigations, and self-assessment tests.
- **Assessment and feedback:** The feedback is provided by the users themselves (the results of the exams are recorded and direct feedback from users encouraged) and in the form of statistics and reports generated by the system for different target groups such as trainers, trainees or managers of the trainees.

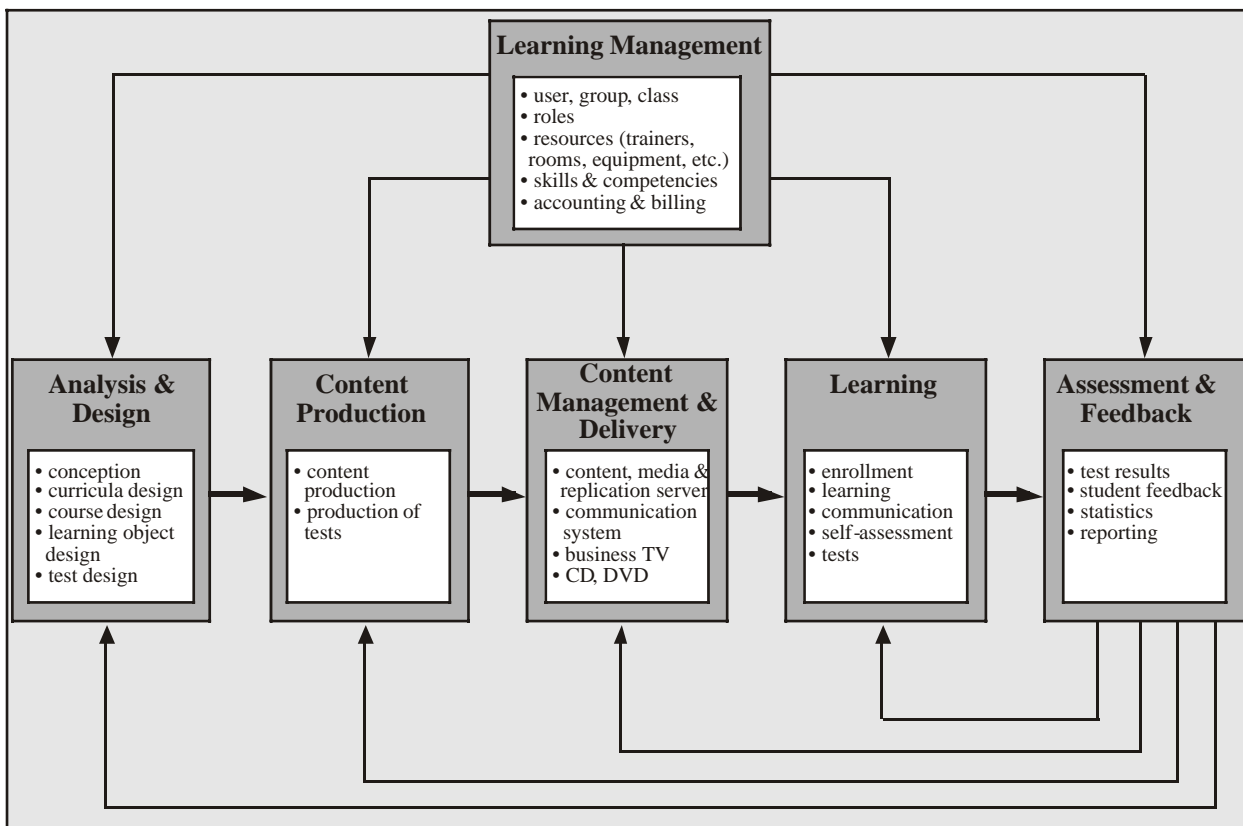


Figure 4.1: Process Model

The following sections describe the process phases and systems needed to support them in more detail. The content management and delivery and its requirements will be specified in a separate section, because this thesis focuses on and provides solutions for this phase of the training process.

4.1.1 Learning Management

The learning management system acts as an umbrella for the previously mentioned kinds of training and covers the overall process. It includes tasks like the maintenance and coordination of the several systems used in the training and the management of the users involved in the learning process.

With the enrollment of a new user, a profile for this user is created and stored. This profile contains information about the user: name, address, company, qualification, skills, competencies, etc. Also, the required login information and those personal files and entries the user wants to save during the learning process are stored. Typically, the user, group, and class management deal with different user roles. Course participants, system administrators, tutors, and other key persons can be identified as potential roles.

Information technology is a rapidly changing field, and prospects for the development of new solutions are booming. This is generally valid for the field of distance teaching and distance learning. In particular, the system must enable adaptation to innovations by ensuring that individual system components can be replaced by newly realized ones. Of course, the system must be extensible. It is foreseeable today that enhanced components will be developed in the future for use in WBT. Even if they are currently not parts of the system, upgrading by adding them to the system should be an essential criterion. Hence, modularity is a desirable feature: the system should consist of different modules that can be added, removed or individually exchanged to meet changing requirements.

As with every commercial training program at DaimlerChrysler, costs for the courses are also defined. So, management should provide the functionality of billing as per the course participation. There are a variety of different billing models: some consider the course as one unit and charge for that course, some split the course into several units (e.g. delivering asynchronous material, synchronous communication with the tutor, taking exams, etc.) and define different prices for them, others count the access time, still others the number of accesses. Billing may be done for a certain time period or in the form of a lump sum. Typically, at DaimlerChrysler, course costs are defined per course. Yet, the billing model should also be adaptable to needs.

Requirements

A user, group, and class management system that can be linked to an external user management system such as a directory system and can retrieve the employee information needed for the enrollment of a user in courses has to be integrated. User management contains information about the individual users and their roles in a course and memberships in user groups.

Further required tasks are quality assurance and billing. Depending on the billing model selected, the learning management system must provide the related functionality of billing the course participation and must permit on-line payment.

The system must provide an open architecture, i.e. the different modules (e.g. the authoring component, or the course-structuring tool, etc.) should be exchangeable. Also, the system has to allow new modules to be added and unneeded ones to be removed.

4.1.2 Analysis, Design, and Production of Content

At DaimlerChrysler, new courses are generated and existing courses are changed on the basis of use cases. Initial considerations to be investigated concern the actual aim of the training course. Possible goals are a) the launch of a new product and/or b) training of service technicians and sales personnel using new technologies or c) the general continual process of education (long-term training).

Both training material and any technical information available for a specific topic are used for course generation. Hence, the information has to be available in a form that will enable it to be included in courses. Technical information is continuously input on a media server by authors. With time, more and more information is added, modified, and verified.

Each bit of content is concatenated with meta-data [LTSC00]. This includes the topic of the content, a brief description of the content, the author, keywords, the modification date, the target user group, etc. Also, meta-data may be automatically or semi-automatically generated with the help of meta-data generation tools.

After the data has been produced, all the technical and training information is stored together on the media server. The people responsible for training create the structure of the courses and have to select and to qualify the related information. Using the meta-data stored together with the content data, they have to decide which learner group is to learn which information and to define the time when the information is to be delivered to the learner groups. Also, they determine the type of training to be utilized in order to deliver a specific training content and set the knowledge prerequisites necessary for a certain lesson.

Within a course, the authors define time schedules, a session and chapter structure, and a learning path which are all linked to the related content information. Tests are included for self-assessment, and preliminary and final testing is carried out to assess existing skills and the skills gained during the learning process.

Furthermore, the authors can provide an overall form for the course and the individual lessons. When describing the courses, authors have to be able to define prerequisites (e.g. degree of knowledge) for students wishing to attend the course. This means the course description should reflect learning goals, an overview of the modules and the content, the level of difficulty, the time required for the course, a schedule, and costs.

Requirements

Course authors need an environment in which they can build the course in accordance with a given model. The course should be structurable in several lessons (i.e. sessions) which, in turn, consist of a page structure. The page structure must be organized as hypertext and give learning paths. The system must provide the possibility of stable connections to external databases and the use of database information in the course structure. For the purpose of material structuring and organizing, it should be possible to use meta-data which is stored in the external databases. Furthermore, the system uses its own meta-data. Such meta-data is attached to the course modules and is used to determine whether a user possesses the prescribed degree of knowledge for a course. In addition, using meta-data enables the course structure and content to be individually adapted to the users' preferences and skills.

Not only should the system permit the addition of material, a short interactive tour of the course should be provided. Such a tour would, among other aspects, show how the learning process is structured and organized within the course. There are different formats of material like HTML pages, animations, and simulations. Tests are included in the course structure as well: preliminary tests, finals (results are recorded and user profiles updated), and self-assessments. The system must be able to compare the results of the preliminary testing and the information in the user profile with the system meta-data to determine the user's level of knowledge.

Trainers (may also be authors) have access to the same methods of communication as the learners. They are identified by the system as tutors, and tutor-mail sent to them by learners can be answered from within the system.

4.1.3 Learning

When using the WBT system for learning, the user opens a Web browser and logs into the WBT system. When the user starts the login process, the system retrieves the user profile and compares the login data. Then the user is granted access to the standard learning desktop containing a news center, a notice board, the learner's bistro (chat), the personal learning desktop, the curriculum, FAQ that can be searched for, dictionary of technical expressions, etc. Within the personal learning desktop, users have links to their private course schedule, bookmarks, and notes. Such personal information is stored with the user profile.

The private schedule area displays information that concerns the learning organization of the user. Users can add and edit personal schedule entries or view announcements of special events (e.g. a chat conference on a specific topic). Also, the schedule shows the user's own learning process. This includes the modules worked through, a history of system actions initiated as well as results of

exercises and tests passed. Moreover, the schedule should give an overview of any particular problems the user had trouble with.

The standard learning desktop contains starting points (icons, links) to those communication means accessible. Communication and cooperation tools are crucial for keeping in touch with other students and instructors. Hence, course-related group interaction such as chat (there are text-based and audio-visual chats), course-related discussion forums (this promotes discussions with other course participants in different time zones) as well as tutor-mail and learner-mail are required. These communication tools utilize the user and group management functionality. Users can join a group of members with the same interests and contact them directly by addressing the whole group.

Within the area of communications, there is a need for some special means that would permit users to make arrangements with the tutor. Some of the auxiliary means deployed here could be, for instance, awareness- and arrangement-oriented schedules or timetables.

Within the curriculum, which can also be accessed through the standard learning desktop, an overview of the syllabus is given and the structure of the whole course including the parts that take place using other training methods (e.g. Akubis, CBT, etc.) provided. The WBT course structure consists of individual sessions, which are also called lessons. In the course structure area, in turn, general information about the aim of the WBT course content is given. A help system contains information on handling the system and organizing the learning process.

Every session starts with a brief introduction of the topic of the lesson and a preliminary test which assesses the user's current level of knowledge of the subject. Hence, the modules to be suggested for review can be decided, if need be. Also, the user will not be admitted to pass this session if the required skills are not gained. Testing is also intended to help refresh and reinforce the knowledge acquired in earlier lessons. The main part of a session is comprised of the new learning content. A final assessment test is given to evaluate the newly gained knowledge and to update the user's profile by adding the new skills acquired during the session.

As already mentioned, the learning content is structured in learning paths (the user is guided through the lesson) and in a hypertext structure (users can freely browse the content and explore it at their own discretion). In addition, the user may decide which learning method to use for a specific session. The content consists of presentations (i.e. static knowledge), animations (i.e. the presentation of dynamic content such as processes), and simulations (interactive part of content; the user can change some input parameters and watch the changes in behavior) with practical exercises (that are also simulated) and self-assessments.

Parallel to reading and learning, the user can make use of any other tools (e.g. communication, FAQ, schedule planner) included in the system. After finishing with a tool, the user continues learning at exactly the same point left when switching over to the other system tool. This ensures that the user does not drop out of the learning process.

Special technical expressions occurring within the course content are linked to a dictionary. Clicking on an expression will open the dictionary, thus allowing the user to read the explanation or definition of the term.

Requirements

The WBT system provides an initial desktop to the courses. This initial desktop should contain personal and public areas which can be created either according to the user's own needs or according to the structure described above.

The users' learning environments should have a connection to their own profiles that contain both all the personal data entered and information about the learning progress.

Communication tools must be available: at least a chat tool, discussion forums, videoconferencing, and e-mail. These tools contain communication channels relating to specific courses or topics of interest to the user groups participating in these courses.

The course structures have to be displayed in form of a schedule, a hypertext structure, and a learning path. The material is displayed within the system, e.g. the Web browser. Parallel to the self-learning process, any of the system functions available can be accessed without the user losing the current learning status.

4.1.4 Assessment and Feedback

The learning content consists of some kind of content presentation together with practical exercises and self-assessment tests. The test results are displayed only to the learners themselves who can store the results in their own private notes areas. This information is not stored centrally in the system.

Every lesson ends with a final test. During the finals, all communication means and the links to all training material are automatically released so that the information tested cannot be accessed. Depending on the test results, the system tells the user to continue with the next lesson or advises that certain chapters be repeated. The test results can be saved in the private notes area and are used to update the user's profile. Also, the qualification profile of the user is adapted to add the newly gained skills.

After the user has passed all the online courses required for a certain training step, the system informs the user's superior and the company about the results. The user is then allowed to take the next training step.

Requirements

The WBT system provides tests that are automatically evaluated. The results should be made available to the user. For certain test types (finals), the results are recorded. The evaluation of the recorded test results is used to assess the existing knowledge and the know-how acquired during the learning process.

The system should be able to automatically generate reports based on the updated profile. These reports are sent by e-mail to the learner's superior and the company for incorporation into their own records.

4.1.5 Roles in the Training Development Process

The greatest advantage of WBT is that the system supports several forms of learning, including the different learning styles: self-study, group learning, and seminars. For this purpose, the system provides management utilities that enable the administration of user and group profiles.

The system contains user and group management which generates a profile for each user. This profile contains information about the user's name, contact information, working situation, qualifications, etc.

The learners, who are the actual target group of the distance learning system, use the learning environment as described in the phase learning. Among other things, their profiles reflect whether they have already taken other courses and it ensures that users have the knowledge required to start a certain course. User profiles also contain information about training results and are automatically updated every time the learner passes an exam.

To realize the learning environment, the tutor works together with the author. In some cases, one person could play both roles. But, typically, the tutor is responsible solely for the course conception

and course instruction. The author creates the course material according to the concept submitted by the tutor. Authors choose the media and fill it with content corresponding to the guideline.

The last key actor in the development process of distance training is the administrator. The administrator is responsible for maintenance of the system, provides all the components needed for training, and works together with other administrative personnel.

In addition to delivering the related user/user group management functionality, the system has to provide the functionality required to support the above key players in the training process. This includes, for example, authoring capabilities for the authors, a tool for the purpose of analysis and design of courses, and, for tutors, a teaching environment that is connected to the learning environment. Also, the administrator requires a management environment enabling administration tasks to be executed, and the learner needs an effective learning environment.

4.2 Content Management and Delivery

As already pointed out, this section especially highlights the content management and delivery and parts of the content production phase of the learning phase. Figure 4.2 shows the major components of the teletraining and distance learning system as it now exists or will be deployed for the distribution of information and training sessions or learning objects. The system is used for the training of sales and service personnel in the passenger car and commercial vehicles divisions within the DaimlerChrysler Corporation. The system approach consists of the following parts:

The satellite-based system uses Digital Video Broadcast (DVB) technology. The audio and video streams are MPEG-2 encoded. The training sessions are transmitted from the corporate TV studio to several locations all over Europe. In the studio, there is a variety of equipment for multiplexing MPEG-2 streams, modulating them, and uplinking them to the satellite. Each uplink channel has a capacity of 8 Mbps, yet the live video only uses 6 Mbps. So, the remaining 2 Mbps could be used for data.

On the other side of the satellite system are several receivers distributed all over Europe. The participants can be present at the training sessions and may ask questions at preset times. Multiple ISDN B-channels (6 or 12) are used as interactive feedback channels from the trainees to the trainers in the studio.

The combination of satellite and ISDN communication is currently the most effective way of achieving both high-quality video transmission at low cost and interactive feedback channels at moderate ISDN tariffs. How this system can be extended to North America through the transmission of MPEG-2 transport streams via native ATM has been published in [Ebe97] and [ERS97a].

These parts of the system have been successfully deployed for three years now. The system is used approximately 150 days per year. The other components are currently in the development stage. However, some have reached the testing and trial phase.

The central video and media server is used to store both training sessions and all kinds of training material. In addition, the material may be edited subsequently. Hence, special workstations which are dimensioned for this kind of work have been set up for the trainers and content managers. The processed material may be used afterwards for the training sessions, e.g. the trainer can play in the answer to a frequently-asked question.

In addition to the training sessions described above, the question of individual training has to be addressed. Nowadays not only literature but also computer-based trainings (CBT) are becoming increasingly important. The content of the CBTs is distributed on CD ROMs. In the near future, Web-based training (WBT) will play an even more significant role. As a result, a need for replication servers at several locations is anticipated. For administrative purposes, the original contents would have to be available on the central media server. Caching strategies for the

replication server must consider the status of the trainees. However, this shall not be the focus of this work.

A connection to the Intranet, Extranet, and Internet is crucial for the hotline, for marketing strategies, and to support the individual trainees.

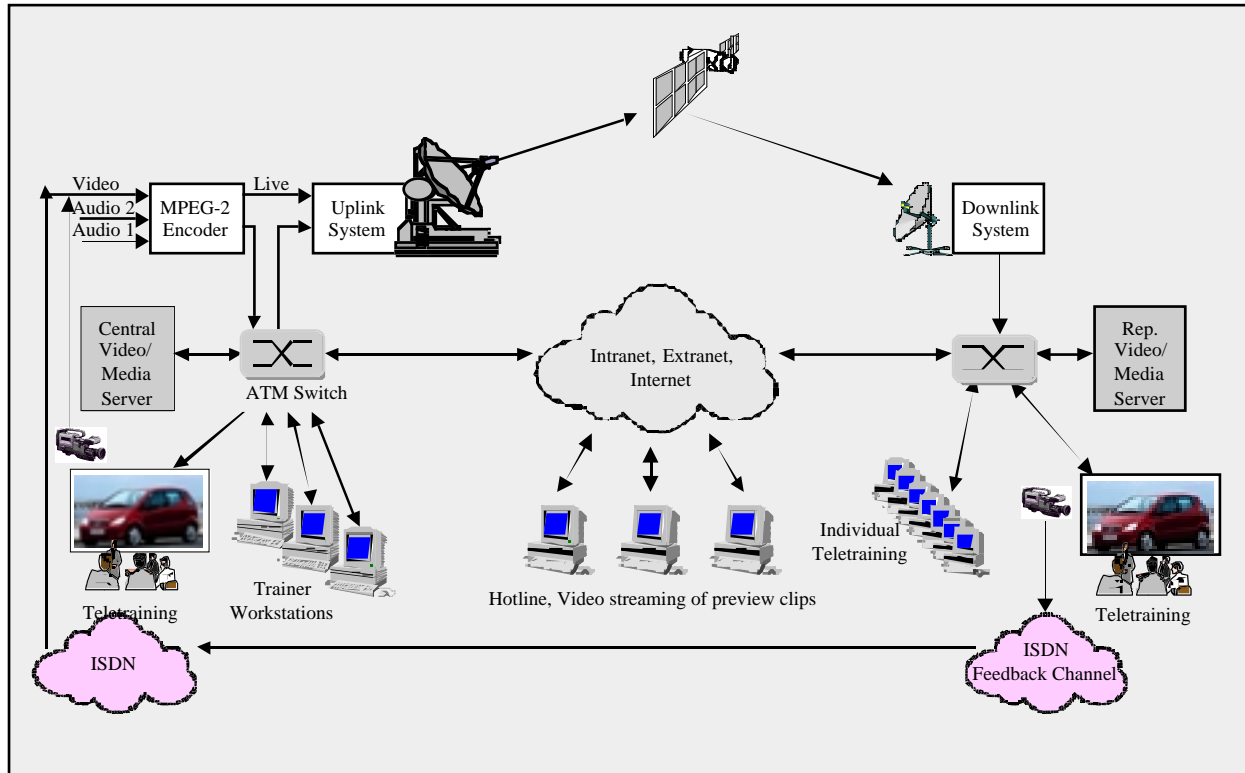


Figure 4.2: Teletraining and Distance Learning Scenario

In the above-outlined scenario, four types of communication channels are available [MeEbe99]:

The satellite channel is a high-quality, one-way road that enables efficient distribution of the current training session. A possible data channel also enables the distribution of bandwidth-intensive data such as video and teaching material.

The IP network (Intranet, Extranet, and Internet) provides whole-time connectivity. However, this channel is not capable of delivering QoS with the exception of best-effort service. Although this is not a medium which is well suited for the transmission of high-quality video, it could still be used for a Web- and chat-based hotline and to inform the participants of new trainings by means of low-quality preview clips.

For cost improvement reasons, the ISDN feedback channel is established solely during the times feedback is to be provided. ISDN provides a high-quality channel on demand.

In the locations where group teaching takes place and, in particular, in the studio, ATM is the ideal high-quality communication system. ATM provides enough bandwidth and enables prioritization of time-critical traffic over less time-critical traffic. Therefore, during a training session, other trainers can simultaneously prepare their teaching materials or answer questions on the hotline without the danger of the session breaking down.

A further essential aspect to be considered is the requirements of the participants. As their needs differ very widely, they can be divided into five groups:

Trainers use the material on the media server to prepare a session, then utilize the material during the session, and evaluate a training session afterwards. Usually, transactions are bandwidth intensive. Therefore, trainers need a direct high-bandwidth connection to the central media server. To avoid consistency problems, all of the trainers have to be connected to one logical server. Although the trainers are located at the studio site for this purpose nowadays, in the near future they might be able to do this from their offices. Consequently, there is a pressing need for high-capacity access networks.

The trainers and moderators of the live-broadcast training sessions require a high-quality channel to all of the locations the sessions are to be transmitted to. This is done via the satellite channel. On the other hand, these trainers need high-quality access to the media server on demand. Also, it is likely that the trainers would receive on-line questions from the Web. It is often beneficial to answer some questions live and not just in retrospect after a session.

The participants in a live session have to be permitted to ask questions. Moreover, the questions put have to be transmitted to all the other participants at the other locations, so that all of the participants can profit from the exchange of knowledge. In principle, although it would be sufficient merely to transmit the voice of the questioner, studies show that communication is more effective if it is effected when people see each other. As the quality of such a video transmission is not of such major importance as during a teaching session, six or 12 ISDN B-channels with H.263 encoded video would suffice.

The individual training participants need access to training material. In addition to the distribution of CD ROMs, Web access extends the benefits achieved. The advantages are faster feedback, better control of the trainees by the trainer and, thus, enhanced support. Not only is there a need for textual and pictorial information, but provision of video information is gaining in significance. Ideally, people should be able to send and receive information from their normal desktop computers. Yet, it has to be taken into consideration that their high bandwidth requirements must not impact the work of other users in the same network. In addition, access to the central server takes place over the Intranet, which could cause a bottleneck. A distributed server architecture can help here.

The last group consists of users whose only concern is getting the information needed. They are not interested in high-quality video. A short low-quality preview clip which provides an informational overview of the session easily fulfills their requirements.

In summary, there are three major requirements for QoS from an application perspective:

First, the editing and management of training material in high-quality MPEG-2 encoded audio, video, and related multimedia information needs a.) reliable transmission between content processing client and server to remain the high quality and b.) hardware decoder and external high resolution monitor to display videos in the best quality. The roles involved are content editors and managers, trainers, and system administrators.

Second, the delivery of live training sessions via the business TV-based teletraining requires a.) again high-quality MPEG-2 encoded audio, video, and related multimedia information to ensure that even small details and fast movements of vehicles are displayed in best quality, and b.) large-scale projection at the receiver side with hardware decoder for the same reason. The audiences are in specially equipped classrooms.

Third, with respect to Web-based training, only the provision of low-quality encoded audio and video (e.g. H.263, RealMedia, Quicktime) via Intranet or Extranet can be delivered to the dealership in the near-term future. The main reason are still the high costs of connecting the dealerships to the core network of the company.

Fourth, in the case of delivering low-quality video over the Intranet or Extranet, standard PCs with software decoders can be used at office workplaces or specially equipped rooms for mechanics.

For some of the core elements of the learning system – the content production, management, retrieval and delivery system – there are a total of seven groups of actors at the system boundaries: trainers, trainees in classrooms, trainees in their offices, content managers, content provider, content editors, and system administrators.

The roles of the trainers have to be further broken down according to the type of training they are responsible for: either classroom-based, teletraining, CBT or WBT. The trainers have to access the system for preparation, teaching, evaluation, and assessment of courses. Also, they have to support the content manager in the editing of teaching videos and new training materials in cooperation with the content manager.

Additionally, the user access for both trainers and end-users has to allow them to search for and order training material, permit on-line viewing and previewing (depends on the content and the communication infrastructure), include search mechanisms, such as a tree-like search index, as well as search for keywords and extended search and a billing mechanism.

The content managers tasks are various: content editing or order to a content service center, simple video cut, putting videos together in collaboration with instructors, administration of the content catalogue with the different categories and subcategories, and editing of the on-line news magazine.

The tasks of the system administrators are to store and delete content, add and delete content-specific information, monitor system access, report user statistics, data backup, retrieval, and long-term archiving, and hardware and software administration.

There are numerous services which are required by these actors at the system boundaries: the video streaming from source objects (e.g. video encoder, video server) to sink objects (video decoder, video servers) in different quality-of-service levels. In order to satisfy user demands as well as to cope with the end-to-end communication system, quality of service has to support the most often-used video compression techniques such as MPEG-2 encoded transport stream (6 – 15 Mbps) for high-quality video, MPEG-1 encoded system stream (1.5 – 2.5 Mbps) for medium-quality video, and Real G2, Quicktime 4, and H.263 for low-quality video. The encoding and transcoding service has to control encoding and transcoding devices and processes and associated devices such as digital video recorders to produce the requested audio/video quality. A security service can be used to add watermarking to videos for copyright protection. The accounting and billing service of the media service has to be linked to the corresponding service of the learning management system. The database service is required for the storage and retrieval of media and learning object meta-data. The search engine makes use of the meta-data to retrieve media such as videos or animations. The Web service is used for a consistent interface to the system.

These services have to be either loosely coupled, as is the case for the accounting service which, for example, has to monitor the access to the various system parts, or tightly coupled, e.g. between the database and the video assets.

4.3 Summary

In summary, the system requirements of a heavily distributed multimedia application such as the distance learning application presented are the following: First, the solution has to support a heterogeneous network infrastructure (e.g. satellite, ATM, Intranet, xDSL) with QoS support and multicasting. Second, a variety of multimedia devices such as audio/video encoders, decoders, and servers have to be seamlessly integrated. Third, heterogeneous server equipment, e.g. video/media server, Web server, DBMS for video meta data, Web-based training system, etc. are parts of the entire system. Fourth, the system has to be capable of coping with additional services such as accounting, billing, directory, and transcoding services. Fifth, a heterogeneous client equipment with respect to operating systems, hardware performance, memory, etc. is commonplace and has to be supported.

As stated in chapters 1 and 2, the focus of this thesis work lies in the definition of a consistent framework for the control and management of multimedia transport streams through object-based distributed system technology and in binding the various services described above seamlessly with the same architectural approach. Hence, only the relevant parts of the whole system will be discussed further.

5 Management Architecture

As analyzed in the previous chapter, the requirements for a quality-of-service management architecture arise from the process chain which runs from the production to the consumption of learning materials. The original material is produced in an extremely high quality. It is to be offered to the consumer in different qualities and through different media, depending on the kind and purpose of the learning, personalization, cost, and technical constraints. Additional services such as accounting and billing have to be included, as they are related to a service process.

Text, graphics, pictures, (3D) animations, virtual reality, audio, and video sequences are used. The following communication and storage media are eligible alternatives in the process chain: satellite, high-speed communication networks (ATM, Fast Ethernet), and Intranet (heterogeneous network) as communication media and CD ROM, VHS tapes, (Digital) Betacam, and DVD as tertiary storage media for archiving and delivery.

Within the context of this thesis, the media high-speed communication networks and Intranets are of special interest and will be investigated in more detail. As a high-speed communication system, ATM is the best choice in the content production area, since ATM technology is currently, and will in the foreseeable future continue to be, the only technology which is able to provide the QoS needed for the transmission of high-quality continuous data such as audio and video. Typically, the audio-visual content is encoded in an MPEG-2 transport stream and stored in the media server. The generation of lower-quality audio and video and other kinds of content are made available either through an off-line batch transcoding service or by re-encoding the same sequences in another lower-quality compression format such as MPEG-1 or Quicktime 4.

The different approaches described in chapter 3 show that there is not just a single solution for providing a communication and middleware architecture which supports end-to-end quality of service and for seamlessly binding application services with the same architectural approach. The requirements set out in the previous chapters have to be kept in mind:

- Heterogeneous network infrastructure (e.g. satellite, ATM, Intranet).
- Heterogeneous server equipment (e.g. video server, Web server, database for video meta data, operating systems, etc.).
- Seamless integration of additional services (e.g. accounting and billing, transcoding of audio-visual data, etc.).
- Heterogeneous client equipment (e.g. operating systems, performance, memory, etc.).
- Scalability and the use of open standards or de-facto standards.

5.1 System Architecture

This thesis proposes CORBA [OMG95] as the architecture of choice to glue the different parts together on the client and server sides. By separating the transport stream from the control stream, this approach can make use of the special features of the communication infrastructure (e.g. ATM's QoS, widespread Intranet usage) and utilize CORBA for the control stream. As described in chapter 3, there is a consensus that control and management of audio and video streams should be separated from the streams themselves [MGRO97] [ScRaLa98]. However, the relevant papers show that proposals for a QoS management architecture are still immature or cover only parts of the universe of discourse. Figure 5.1 shows the system architecture with the separation of the real-time transport system from the CORBA-based middleware infrastructure [EbRu99]. The overall

architectural approach for the training online system as described in the previous chapter is specified in [RuWoEbHo01].

CORBA is employed as a middleware system to connect the system parts on both the client and the server sides. On the server side, CORBA is ideal for building a distributed system that can easily be distributed on different computer hardware, for instance, for load-sharing purposes. To a large degree, the complexity of the distribution can be hidden from the application itself. Hence, the application can be effectively structured, and functionally-independent parts of the system such as the billing service can be encapsulated and possibly distributed. Access to legacy systems or the integration of database management systems (DBMS) can be achieved easily and efficiently [ERSS97]. Client platform independence may be achieved by using Java on the client side with the Java to CORBA language mapping and Java media players [Sun97]. This is, in particular, true if CORBA is implemented in Java itself. Yet, it remains to be investigated whether a Java-based ORB on the server would be sufficiently performant.

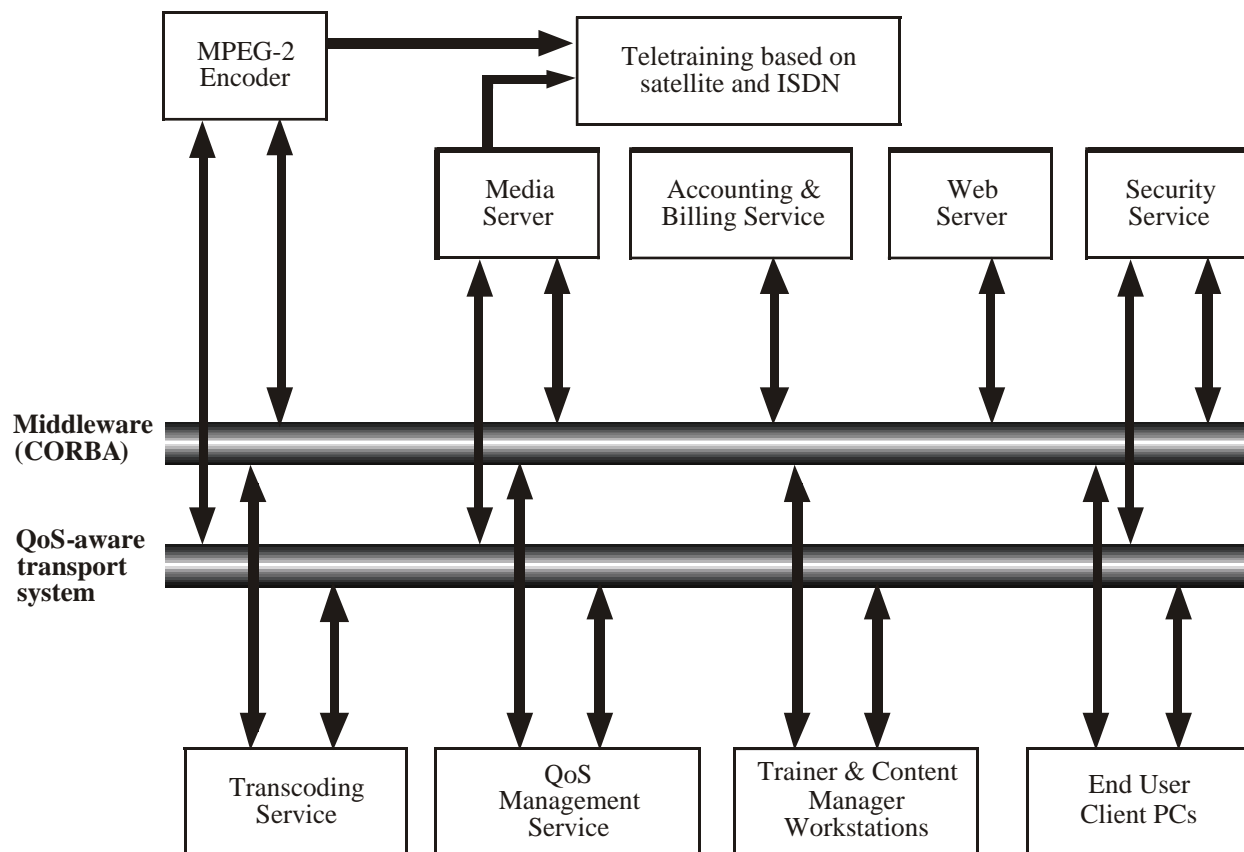


Figure 5.1: System Architecture

5.2 Multimedia Device and Stream Management Architecture

The control and management of continuous media streams represents one crucial aspect of the entire system. As stated in section 3.3.1.1, the Object Management Group has developed a specification for the control and management of audio and video streams [MGRO97]. A few architectural concepts of the proposed architecture are derived from this CORBA telecoms specification. The architecture proposed yet has the added benefit of reducing the complexity of the specification, on the one hand, and extending it, on the other hand, to fit the requirements as needed [Ebe98a].

The QoS management service has a special significance with respect to efforts aimed at achieving acceptance for the whole system by the user. QoS management stands for control, enforcement, monitoring, and mapping of QoS features both as service user and service provider at the different levels of abstraction. QoS mapping describes rules for the translation of a QoS specification from one level of abstraction to another level.

One basic assumption underlying this hypothesis is that the individual parts of the system are separated functionally from each other and encapsulated in the object-oriented meaning. Interfaces are defined by means of the CORBA IDL (Interface Definition Language), whenever remote access to a component is necessary. Typically, access to system resources such as the operating and communication systems is solely possible via the APIs cited in section 2.4. Application programming interfaces (API) with QoS support at different levels of abstraction and the mapping between different levels are a vital architectural element. Since these APIs often reside at a very low level of abstraction and, in addition, are dependent on the operating and communication systems, this thesis proposes a system-independent API. This approach can be smoothly integrated as an essential element in the multimedia device and stream management architecture, because it fulfills both prerequisites: interface definition with IDL and implementation with Java [ERM99].

Figure 5.2 depicts the multimedia device and stream management architecture and again shows the separation of the media stream and its control in detail. A unidirectional transport stream with two endpoints – one acting as source and the other as sink objects – is depicted. The sources, sinks, and flow endpoints are controlled via the management interface objects. The interface management objects communicate with the management service via the Basic Object Adaptor (BOA) of CORBA. All of the interfaces between the objects have to be well defined by a notation such as the Unified Modeling Language (UML) [Oest98] and an interface definition language such as OMG's IDL.

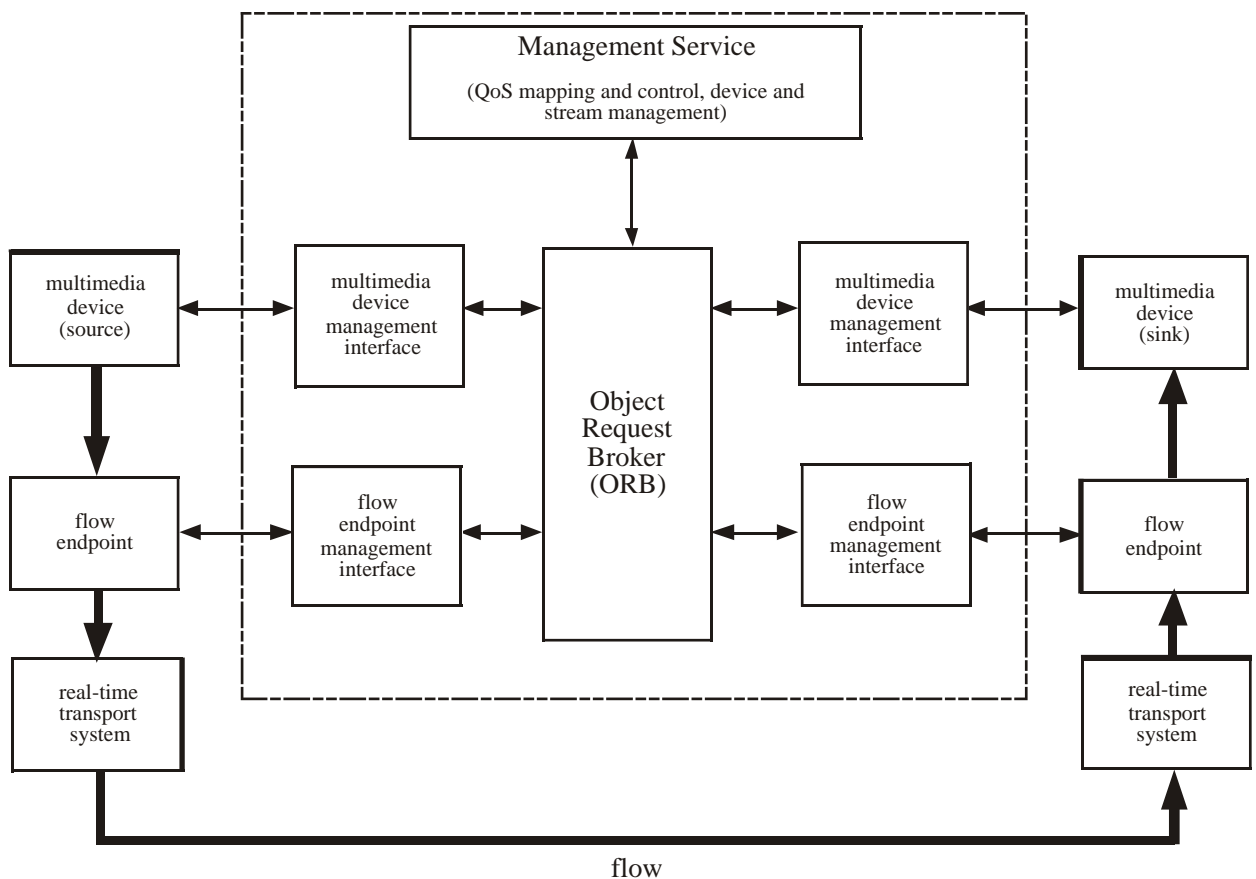


Figure 5.2: Multimedia Device and Stream Management Architecture

With this base architecture in mind, the following definitions will be used throughout the remaining parts of this thesis.

A base device (BaseDevice) is an abstraction representing all those objects in the system which are able to receive or send continuous sequences of data. Although any type of long-term continuous data exchange could flow between two objects, this thesis focuses on multimedia applications with QoS constraints.

A flow is a unidirectional and continuous sequence of data, e.g. frames, packets or cells depending on the transport system used. The direction of the flow is clearly identified. Flows terminate in flow endpoints (FEP) on each side of the system.

A stream is an aggregation of one or more flows. A stream is an abstraction of a continuous transfer of multimedia data between two stream endpoints (SEP).

A multimedia device (MMDevice) is bound to a flow endpoint acting as a representation of a source, sink or filter object of a multimedia flow. Typical examples of multimedia devices are hardware or software A/V encoders, A/V decoders and transcoders, media servers, and filters.

Each of these representations is made accessible through its management interfaces and related management objects within the QoS Management Service.

The grouping of flows and flow endpoints into a stream and a stream endpoint is done if an application uses more than one multimedia device as sources and sinks to achieve the functionality of the application. The notion of streams and stream endpoints is made available to applications by the management service in order to manage sets of multimedia devices and their corresponding flows and flow endpoints. If there is only one flow per stream, the terms *stream* and *flow* and *flow* and *stream endpoint* have the same meaning. This is especially true in multimedia streaming applications such as the streaming of a video from a media server to decoding devices in classrooms.

Figure 5.2 shows a multimedia flow, which is represented as a flow between two flow endpoints. The flow endpoints are responsible for setting up and tearing down the required network connection. They have to push or pull data in or out of the multimedia devices to which they are bound. They are responsible for sending or receiving data over a communication system. One multimedia device acts as source of the data and the other as the sink of the data flow.

The management interface objects of both the multimedia devices and the flow endpoints export an IDL interface. Via these interfaces, the objects inside the management service are able to perform management operations on the managed objects, e.g. establishing and monitoring an end-to-end flow between two multimedia devices. These management operations pass through the IIOP-path of the ORB, which is demarcated by the dashed rectangle, as opposed to the data flow, which uses an out-of-band flow. This out-of-band flow of multimedia data can be implemented using protocols and communication systems that are more suitable for continuous data than IIOP. Maintaining this separation between management and transport of a flow is crucial if high performance is to be achieved while preserving the benefits of using a standardized distributed object-computing environment as previously outlined in this work.

5.3 Architectural Integration of Real-Time Transport Systems

My experience with RSVP and native ATM APIs has shown that their level of abstraction is not high enough for application programmers. Therefore, I propose an open approach to support the most popular lower-layer QoS-aware communication mechanisms like ATM and RSVP in a platform-independent manner. The Java initiative of Sun Microsystems has gained a great deal of momentum in the past three years. The Java Development Kit (JDK) and related software development tools are available for most operating systems. Using the java.net package, it is quite

easy to deploy the socket paradigm in different operating systems. However, at the time this thesis was written, QoS support just did not exist in the java.net package.

With the Java Media Framework (JMF) API [Sun99], a collection of classes that enable the display and capture of multimedia data with real-time constraints within Java applications and applets is available. In order to provide a distributed JMF, the java.net package has to be extended for QoS support. This thesis suggests a generic QoS-aware Java API with platform-independent support for QoS-aware communication systems or protocols such as ATM and RSVP [Ebe98b], [ERR98].

In the first step, two flavors of the QoS-aware Java API as shown in Figure 5.3 are provided: (1) RSVP-aware and (2) native ATM services-aware. In a second step, a more generic QoS-aware Java API which incorporates both approaches is given. The API should not be built from scratch as suggested in [ATMF97c], [ATMF97d], [ATMF97b]; instead, it should be an extension of the Socket and ServerSocket classes of the java.net package. In JDK 1.1, Socket and ServerSocket are made non-final, which means that subclasses can inherit from Socket/ServerSocket and can override methods from their superclass. In the case of the RSVP-aware Java socket API, these overridden methods would access the native C methods of the platform-specific RSVP API described in section 3.1.5 via the Java Native Interface (JNI). In the case of the Microsoft domain, it would be mapped to WinSock 2 and, in the UNIX domain, to a RAPI implementation. For native ATM services, it would be mapped to WinSock 2 native ATM extensions or UNIX XTI's extensions, accordingly.

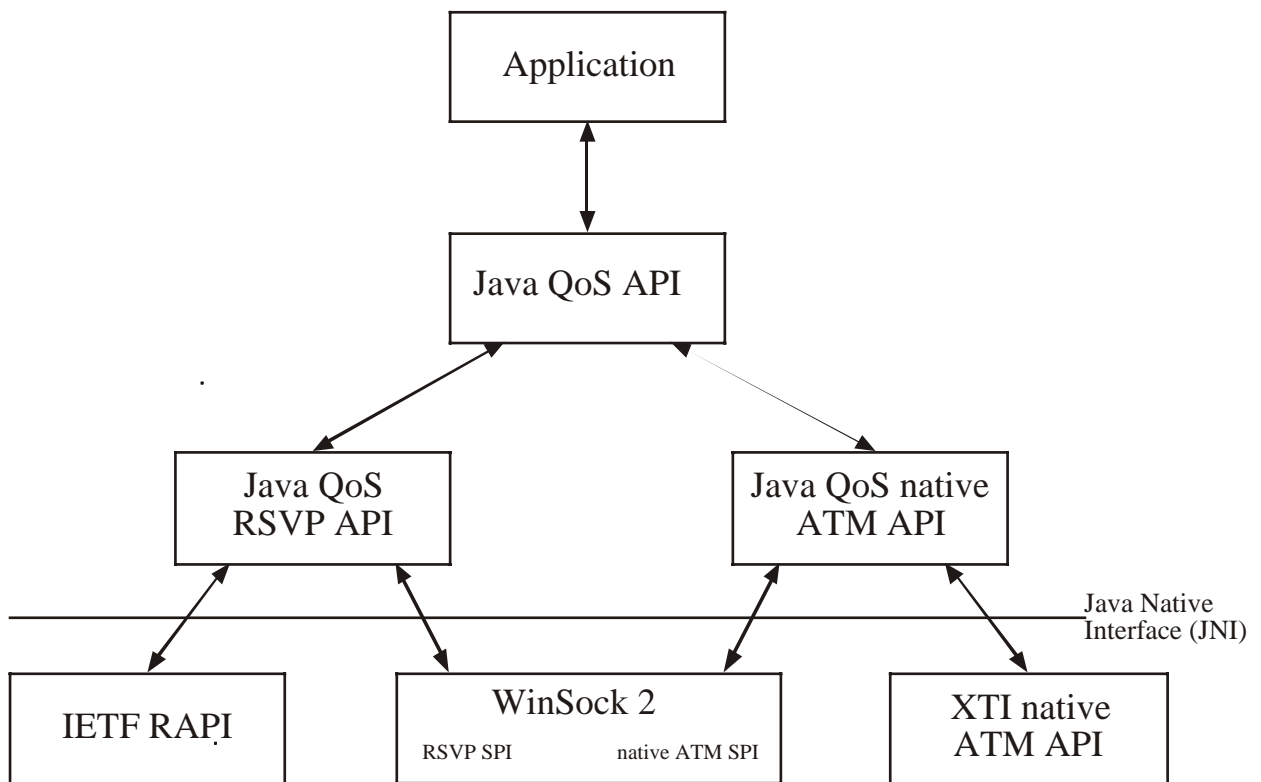


Figure 5.3: Java QoS API

5.4 Integration of Multimedia Devices

The second essential group of devices which have to be seamlessly integrated are the various kinds of multimedia devices (MMDevices) such as audio/video encoding and decoding devices. These devices are either hardware-based like MPEG-2 encoders and decoders or are software-only solutions like MPEG-1, RealVideo or Quicktime encoders and decoders. Because of the diverse nature of these devices, they do not provide a common API. Since this is especially true in the case

of hardware-based solutions, a major concern has to be to abstract as much as possible of the common functionality of these devices at a very high level of abstraction. Therefore, analogously to the previous section, this thesis proposes a layered hierarchy of multimedia device abstractions as shown in the figure below.

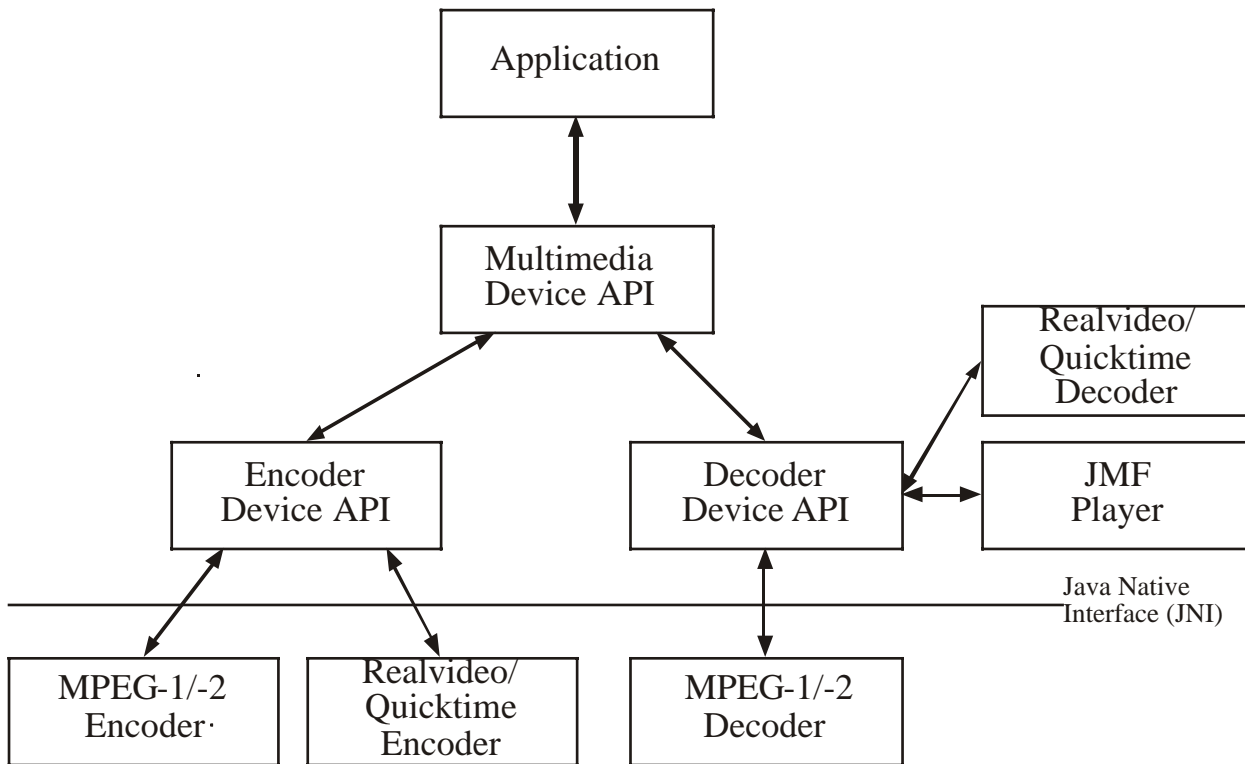


Figure 5.4: Multimedia Device API

At present, the player functionality of the Java Media Framework (JMF) [Sun99] version 2 is only used for low-quality audio and video streams, because it was not able to integrate high-volume A/V streams such as MPEG-2 encoded content efficiently. This kind of high-volume data should be shifted as directly as possible between the related devices. With the publication of the second version of JMF, further investigations will have to be performed to find out whether it would make sense to integrate more of JMF's functionality into the architecture proposed in this thesis. For example, JMF's Plugin Manager could be exploited to integrate all kinds of low-video quality codecs. Additional study is required to evidence whether high-quality codecs such as MPEG-2 can also be handled with JMF version 2, because it seems that the openness of the framework with respect to the integration of devices has greatly improved.

5.5 System Interaction

The sequence pointed out in Figure 5.5, Figure 5.6, and Figure 5.7 gives a brief overview of how an end-to-end connection is established using the concepts described in the previous chapters, and how control objects, MMDevices, and FEPs relate to each other. Every network node in the system is equipped with special hardware and software components. In the initial phase of each node, the devices are registered in the management layer, e.g. using the CORBA naming service. Any object which is allowed to use the CORBA naming service is now able to browse through the naming service in order to look for service suppliers and their properties. For example, node 1 registers an audio/video camera and an encoding device as MMDevices and an ATM network interface card with guaranteed constant bit rate service as FEP. In addition, nodes 2, 3, and 4 might register the same network characteristics. Node 2 is an intermediate active network node which registers, for

example, an audio/video filter device capable of reducing the overall data rate and video quality by dropping video frames. Node 3 is a media server and is able to store the audio/video flow in its video server and, at the same time, to decode and display it for monitoring purposes on an external TV monitor. Node 4 is a client system which is capable of receiving and decoding lower-quality video and, thus, registers these capabilities.

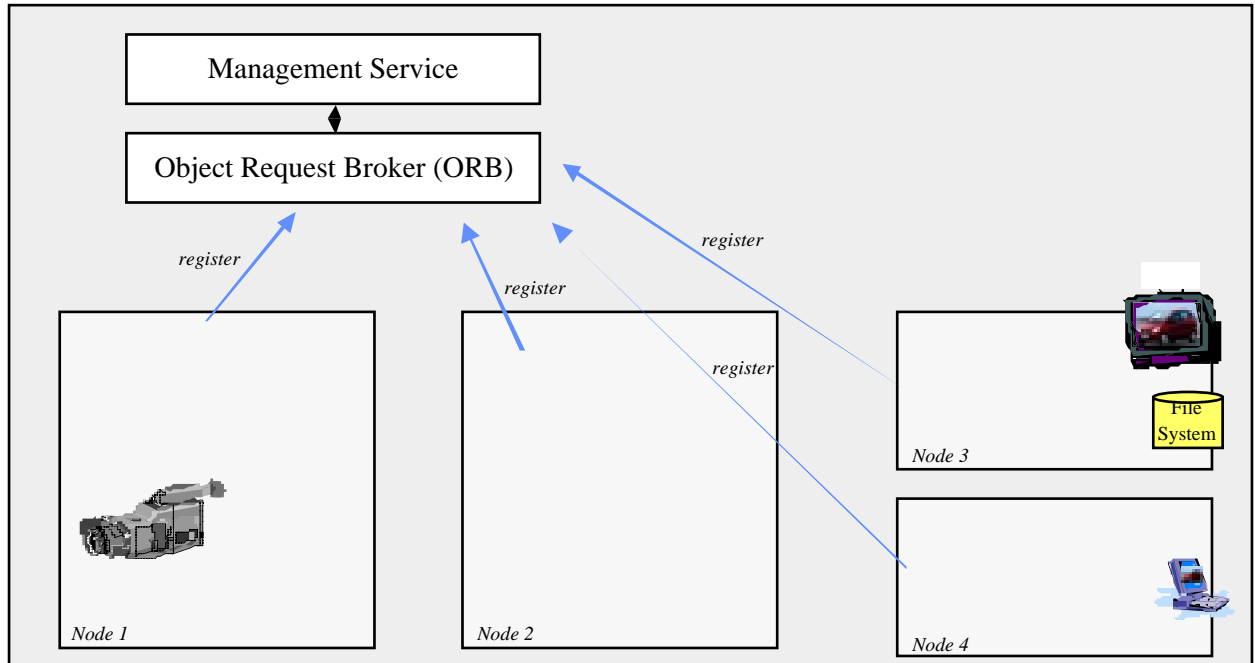


Figure 5.5: Example Configuration of MMDevices and FEPs (Step 1)

An administrator who has to set up a live video broadcast session from a TV studio to remote classrooms and store the session on a video server at the same time can browse the list of available multimedia systems and select the required devices. As shown in step 2 (in Figure 5.6), if a new configuration is to be built up or an established configuration accessed, the application object requests a multimedia device and flow management object. Third, as illustrated in Figure 5.7, the multimedia device and flow management object initiates all the necessary multimedia devices and flow endpoints in the nodes and then starts the transmission.

The multimedia device and flow management object is active as long as the configuration is active, whereas the application object is only needed to initiate or change a configuration. This configuration consists of one audio/video source, three different audio/video sink devices, and one active intermediate device.

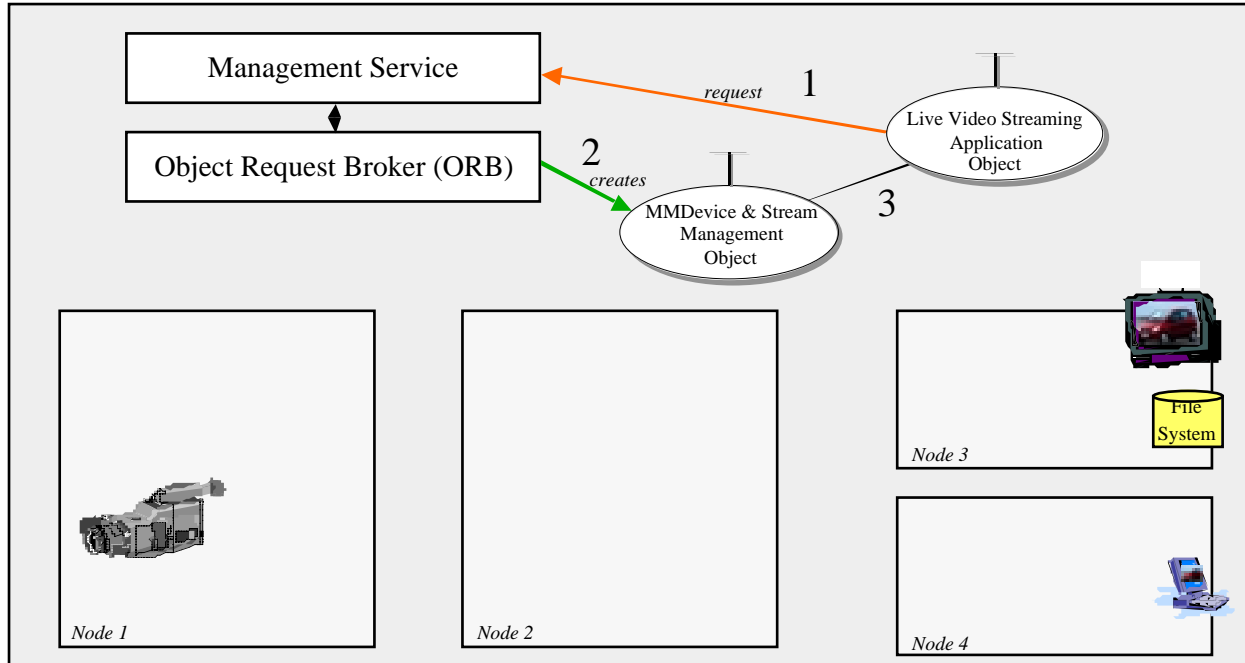


Figure 5.6: Example Configuration of MMDevices and FEPs (Part 2)

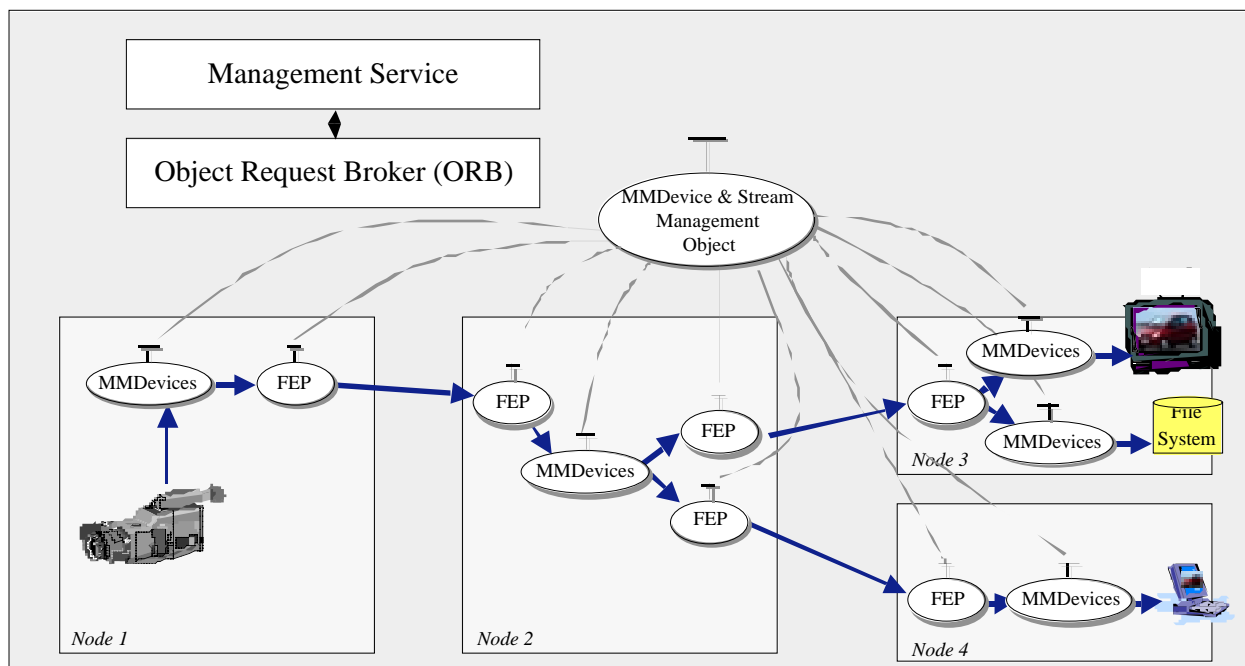


Figure 5.7: Example Configuration of MMDevices and FEPs (Part 3)

5.6 Open Framework

One of the main goals of this thesis is to provide an open and extensible framework where both multimedia devices and communication end systems can be plugged in easily and in a straightforward fashion. The same is to be true for any intermediate devices which receive, process, and transmit a multimedia flow. Typical examples for intermediate devices are filters, transcoders, multiplexers, demultiplexers or encryption devices. These intermediate devices are extensions of MMDevices and will be called Active Nodes (AN). Because of the distributed nature of the architectural approach taken in this thesis, these active nodes can be managed in the same manner as the end-system devices. Furthermore, with the evolution of active networks, this approach can be

used for active networks as well. [DFFR99] emphasizes that the evolution of computer networks towards the active network paradigm strongly depends on the actual benefits that can be obtained by applications. They feel that many of these benefits fall into the following categories: availability of information held by intermediate nodes, data processing capability along the path of a flow, adoption of distributed strategies, and easy development of new network services.

The proposed architecture fulfills all four categories. First, information held by the intermediate nodes can be made available through their management interfaces and described by their registered and exported management interfaces. Second, active nodes as suggested in this thesis have the capability of processing data along a flow of data as set out above. Third, the utilization of object-oriented distributed systems is one of the core foundations of this thesis. Fourth, new network services can be developed rapidly and deployed by the management service through the injection of code. Since another foundation of this work is that Java be used wherever it makes sense, the Java feature of delivering code and executing it on the fly on remote nodes in a secure and distributed environment can be exploited to achieve the fourth category.

Another evolving area comprises the devices which are directly connected to each other through communication systems such as Universal Serial Bus (USB) [USB], Firewire [IEEE1394] [KuWe95] or Storage Area Networks (SANs) [SNIA], [FCIA00]. What these communication systems have in common is that they are able to stream data from a source device directly to a destination device. So, the bottleneck of today's internal computer bus systems can be avoided if a data flow only needs to be transported over a short distance. In the case of Firewire, which is able to transport isochronous data at data rates of 200 and 400 Mbps, it would be well worth investigating whether a direct switchover from ATM to Firewire is viable. A valuable benefit would be that off-the-shelf end systems from the consumer area such as DV (digital video) cameras or hard drives could be directly linked via a Firewire-ATM-Firewire connection. Yet, a proxy manager which is connected to the same communication system as the MMDevices would enable management of these directly interconnected MMDevices. If Sun's initiative Jini is further developed and finds broader acceptance, direct communication with these devices, i.e. without the need for a proxy manager, is conceivable.

5.7 Application Areas and Summary

A key feature of the multimedia stream and control architecture is that it is generally suitable for a wide and diverse range of applications:

- Audio/video entertainment and information systems in all kinds of transportation system such as airplanes, passenger trains, passenger cars, and buses.
- Distance learning and training systems with multimedia content.
- Organization-wide news and information systems with multimedia content.
- Multimedia content creation and production.

These application areas have one characteristic in common: multimedia data has to be streamed from multiple source to multiple sink devices. For example, in the first field of application in airplanes, multimedia entertainment and information data has to be streamed from source devices such as media servers, live cameras, and satellites to sink devices such as passenger in-seat displays and flight attendant panels. With only minor changes, the architecture can be adapted to passenger trains and buses, as well. For future application scenarios such as direct car-to-car communication video communication, the general architectural approach is useful with adaptation to this ad-hoc networking and service scenario.

In summary, it can be stated that the multimedia stream and control architecture can be generally applied to a variety of application environments.

6 Multimedia Device and Stream Control

6.1 Architecture

Figure 6.1 illustrates the multimedia device and stream control architecture. The management layer includes the device control and quality-of-service management system. Because CORBA is deployed as a middleware infrastructure, this part of the application can easily be distributed without increasing the complexity. CORBA is used to separate the transport of time-critical data and media streams from data which is not time critical. Hardware independence is achieved by using Java at the client and server sides. One of the major tasks performed by the QoS management system is the transformation of quality-of-service parameters from one abstraction level to another. This is necessary because an end user cannot be penetrated with lower-layer networking details such as the ATM QoS signaling parameters. A further challenge is the support of access to the multiple devices at multiple nodes in the network.

In the device layer, there are two classes of devices:

The multimedia devices are software representations of hardware and software components which are integrated into a node. For example, this could be the representation of a hardware MPEG-2 encoder or a software MPEG-1 decoder or even an access point of the media server. A multimedia device could be a source, a sink or a filter.

Flow endpoints are software representations of network connections. Every two flow endpoints form one network connection and control these network resources.

To achieve hardware independence, the implementations in the device layer are written in Java. For example, the Java Media Framework multimedia devices are used. At some points, hardware independence is not possible either because most of the hardware components used in the system come with C APIs or because processing power has to be saved.

Several transport networks such as ATM, Ethernet, and the satellite uplink can be deployed in the transport layer.

The vertical dashed line on the right-hand side of Figure 6.1 shows that most parts of the management and device control layer can be implemented in Java and only a minor part has to be done in C++ and wrapped with Java classes. It also shows that the control is totally out-of-band of the multimedia flow. These implementation details are described in depth in section 8.4.1.

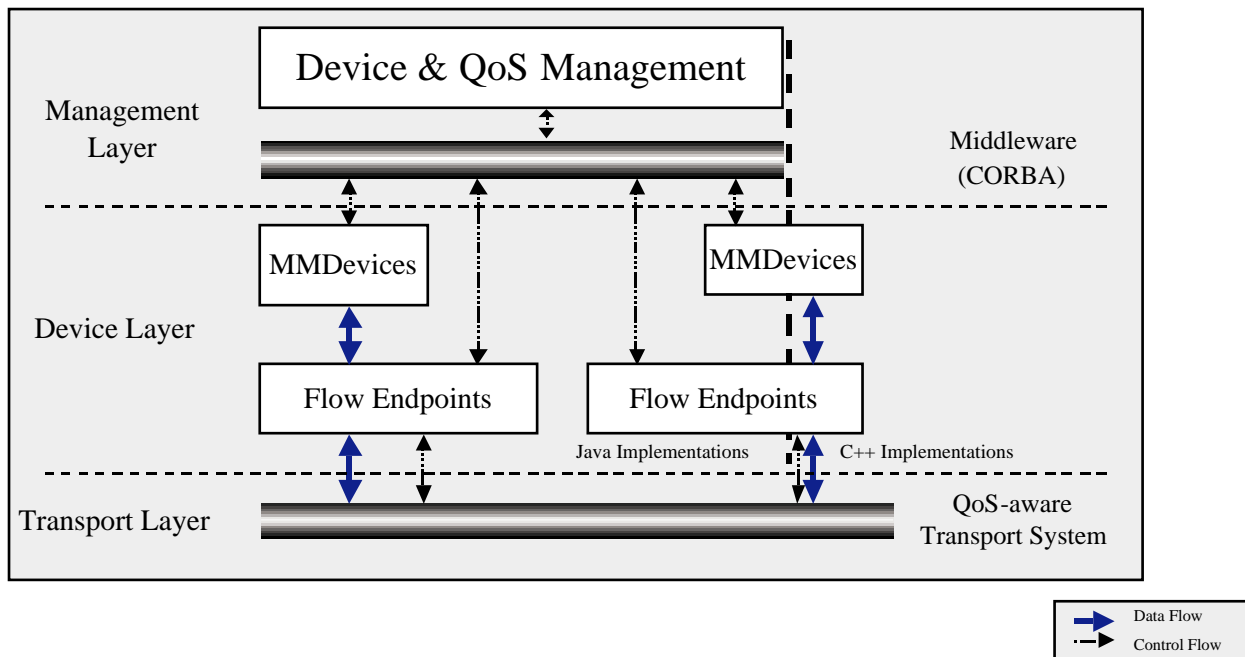


Figure 6.1: Multimedia Device and Stream Control Architecture

A simplified representation of the object model of the device layer is depicted in Figure 6.2. BaseDevice is the object where all the functionalities dealing with binding devices and transporting data from one MMDevice via FEPs to other MMDevices are grouped. The objects MMDevice and FEP are derived from the BaseDevice. MMDevice abstracts all multimedia devices and FEP all network connections.

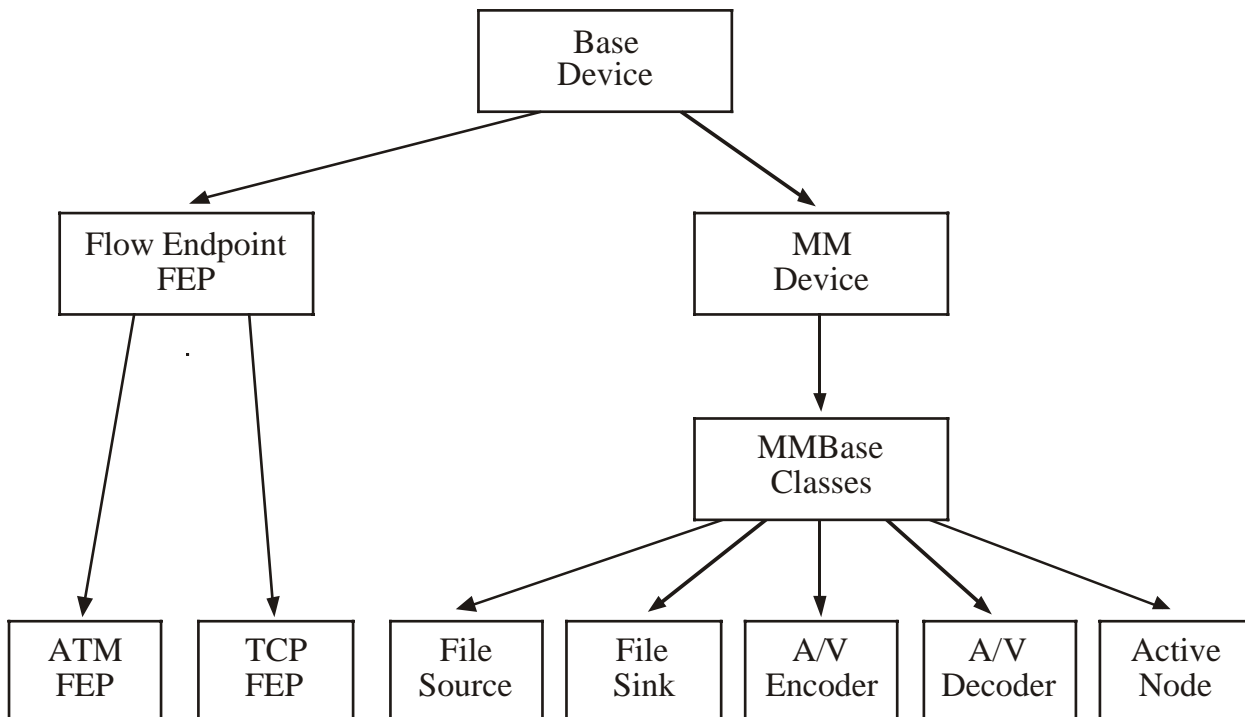


Figure 6.2: Simplified Object Model

MMDevice covers functions which are common for all multimedia devices such as start, stop or pause. The MMBase objects are intermediate objects which hide internal operations such as internal binding or a general interface for read or write access to ring buffers. Finally, the devices themselves such as file sources and sinks, audio/video encoders and decoders, and the active nodes are derived from the MMBase classes. And the different types of network connections such as ATM or TCP are derived from the FEP object, accordingly.

6.2 Communication Model

Base devices (BaseDevice) can be bound with each other in any combination. Every base device can be bound simultaneously to an undefined number of other base devices. The application in conjunction with the device and QoS management service is responsible only for enabling suitable connections in a specific application context. For example, in the application context of this work, i.e. video transmission, the following rules make sense: (1) a source device may only have data outputs, (2) a sink device may only have data inputs, (3) multimedia devices may be bound to flow endpoints and directly to other multimedia devices, e.g. a file source device to a decoder sink device, and, (4) in order to avoid loop conditions, a multimedia device may only have one data input.

Typically, computer-internal communication between devices can be done in one of two modes: (1) data is actively supplied by a source device or (2) data is requested by a sink device. This behavior is called a push or a pull connection between two devices. As depicted in Figure 7.3, each connection between two base devices is associated with the connection parameters: *blocking / non-blocking*, *push / pull*, and *block_size*. The data flow direction results from the assignment of *in* and *out* parameters.

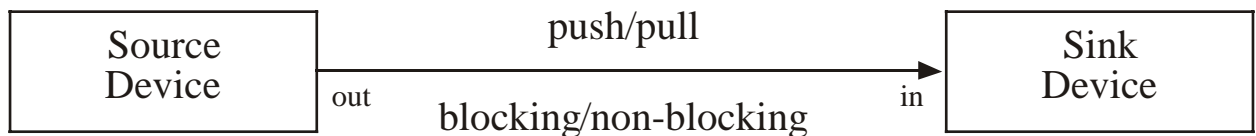


Figure 6.3: Communication Between Base Devices

The operation mode *blocking* defines what happens if data can be received by the sink device as fast as the source device generates the data. The blocking mode of operation is a mechanism to slow the source device down in order to protect data from getting lost. However, in the non-blocking mode, if a source pushes data and the sink is not able to process the data in time, data may be overwritten.

The data flow direction is defined by the binding object, i.e. the device which initiates a data flow. For example, if a source binds with a sink object in push mode, the flow direction is out, because the source delivers data actively to the sink object. And vice versa, if the sink object binds to the source in pull mode, i.e. the sink object requests data, the flow direction is in.

The individual activities of each object depend on the following parameters:

- Blocking push: Source object pushes data into the sink object. If the data cannot be processed immediately by the sink object, the source object waits until the data can be processed by the sink object.
- Non-blocking push: Source object pushes data into the sink object. If the data cannot be processed immediately, it will be lost and the source object continues to work.
- Blocking pull: The sink object pulls data out of the source object. If there is no data to pull, both the source object and the sink object wait until data is available. The source object does not overwrite its buffer until the sink object has pulled all the data.

- Non-blocking pull: The sink object pulls data out of the source object. If a sink object does not pull data in time, the data will be lost. How fast the data will be overwritten depends on the size of the ring buffer and the data rate.

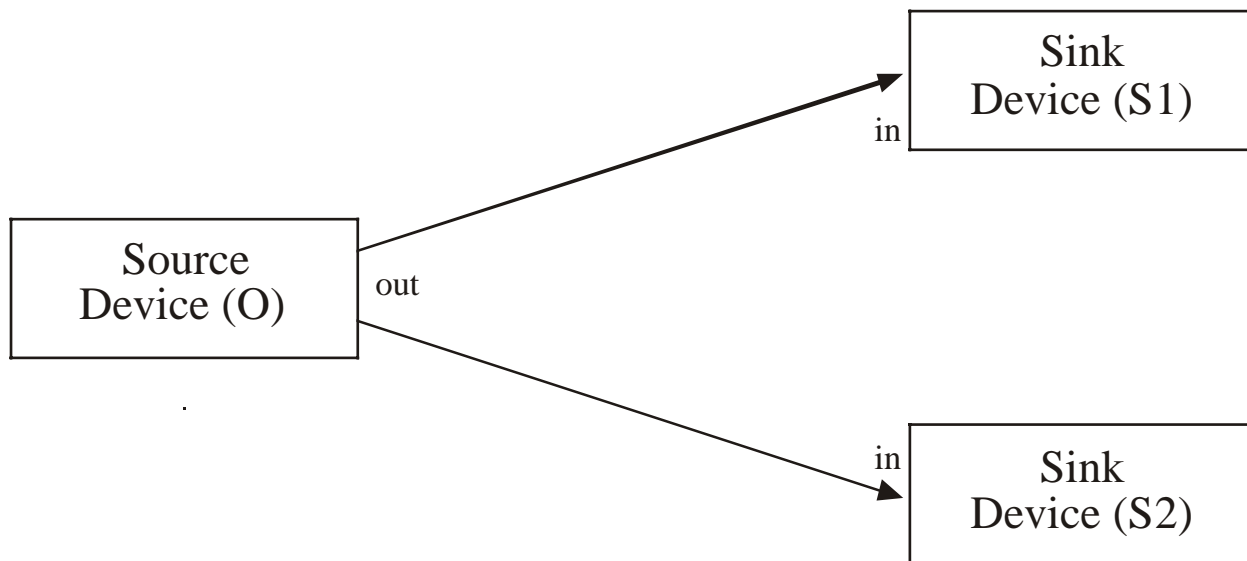


Figure 6.4: Example with One Source and Two Sink Objects

In this case, it is assumed that each sink object is bound to a single source object at the same time, as shown in Figure 6.4. A flow originates from a single source and may end in several destinations. The impact of the communication parameters on the data flow is compared in Table 6.1.

Case	Connection 1	Connection 2	Description
1	push blocking	push blocking	Both sink objects slow down the source object. The data flow in both directions is only as fast as it can be processed by the slowest object. $v_1 = v_2 = \min\{v_{s1}; v_{s2}; v_o\}$
2	push blocking	push non-blocking	Sink 1 slows down the source object to its processing speed. Sink 2 receives data with this rate and loses data if the rate is too high. $v_1 = \min\{v_{s1}; v_o\}; v_2 = \min\{v_1; v_{s2}\}$
3	push non-blocking	push non-blocking	The data flows with the processing speed of the source object. Sink 1 and 2 receive data with this rate and lose data if the rate is too high $v_1 = \min\{v_{s1}; v_o\}; v_2 = \min\{v_{s2}; v_o\}$
4	push blocking	pull blocking	This case corresponds to case 1 $v_1 = v_2 = \min\{v_{s1}; v_{s2}; v_o\}$
5	push blocking	pull non-blocking	Sink 1 slows down the source object to its processing speed. Sink 2 fetches as many data as it is able to process. $v_1 = \min\{v_{s1}; v_o\}; v_2 = \min\{v_1; v_{s2}\}$
6	push non-blocking	pull blocking	This case corresponds to case 2 (sink 1 and 2 exchanged). $v_2 = \min\{v_{s2}; v_o\}; v_1 = \min\{v_2; v_{s1}\}$
7	push non-blocking	pull non-blocking	The data flows with the processing speed of the source object. Sink 1 receives data with this rate and loses data if the rate is too high. Sink 2 fetches as much data as it able to process. $v_{v1} = \min\{v_{s1}; v_o\}; v_2 = \min\{v_{s2}; v_o\}$
8	pull blocking	pull blocking	This case corresponds to case 1. $v_{v1} = v_2 = \min\{v_{s1}; v_{s2}; v_o\}$
9	pull blocking	pull non-blocking	This case corresponds to case 5. $v_1 = \min\{v_{s1}; v_o\}; v_2 = \min\{v_1; v_{s2}\}$
10	pull non-blocking	pull non-blocking	The data flows with the processing speed of the source object. Each sink fetches as much data as it is able to process. $v_{v1} = \min\{v_{s1}; v_o\}; v_2 = \min\{v_{s2}; v_o\}$

Key:

v_1 : flow rate of connection 1
 v_2 : flow rate of connection 2
 v_{s1} : processing speed sink 1
 v_{s2} : processing speed sink 2
 v_o : processing speed source

Table 6.1: Case Study with One Source and Two Sink Base Devices

If the number of sinks is greater than two, then there are four additional cases:

$n = 3$:

{Con.-1(push blocking); Con.-2(push non-blocking); Con.-3(pull blocking)};

{Con.-1(push blocking); Con.-2(push non-blocking; Con.-3(pull non-blocking)}

{Con.-1(push blocking; Con.-2(push blocking); Con.-3(pull n-blocking)}

$n = 4 :$

{Con.-1(push blocking); Con.-2(push non-blocking); Con.-3(pull blocking);
Con.-4(pull non-blocking)};

In general, it is valid that if a blocking connection is used, no data is lost. The source object and all subsequent objects are slowed down to the minimal processing speed of the participating blocking connections, no matter whether pulling or pushing connections are concerned. If, for n connections, connections 1 to i ($i < n$) are blocking and the rest of the connections are non-blocking, then the following is valid:

$$V_o = v_1 = v_2 = \dots = v_i = \min\{v_o, v_{s1}, v_{s2}, \dots, v_{si}\}$$

$$v_j = \min\{v_o, v_{sj}\} \text{ for } i < j \leq n$$

With non-blocking pull and push connections, loss of data occurs if $v_o > v_s$. If $v_o < v_s$ for a pull connection, then the sink objects must be prevented from reading the same block of data twice.

In addition to the blocking and non-blocking modes, a third mode – blocking with time to live – helps to prevent a source object from being blocked too long. Hence, obsolete data can be discarded in the ring buffer and replaced with current data. This is, of course, especially true for live video streaming.

6.3 Object Model

Throughout this thesis, the Unified Modeling Language (UML) [BoJaRu97], [Oest98] is used for formal specification during the analysis and design phases of the software engineering process. UML is the state-of-the-art modeling method which has unified the previous methods of Booch [Booch94], OMT (Object Modeling Technique) by Rumbaugh [RuBlPrEdLo91] and OOSE by Jacobson [JaChJoÖv92]. UML has been adopted by the OMG as the standard modeling language and will be further developed by the OMG. The UML object model in Figure 6.5 highlights the fundamental objects used outside of the dashed rectangle depicted in Figure 5.2. The object model visualized in Figure 6.5 is shown in more detail in appendix 11.6.

The abstract superclass BaseDevice includes the general functionality of all devices. In particular, it provides the operations push and pull as well as bind and unbind.

Push and pull represent the means by which data is shifted from one BaseDevice to another BaseDevice, e.g. from a MMDevice to a FEP. In the case of a FEP which is bound to a source MMDevice, pull means that data is requested by the FEP and the data source remains passive. In the push case, the source MMDevice acts actively and shifts data to the FEP. Obviously, a FEP cannot push into the source MMDevice. Bind and unbind are used to store the binding relationship of the devices.



The subclass FEP (flow endpoint) extends BaseDevice and is used to establish network connections to other FEPs. Multiple transport protocols such as TCP, UDP or ATM are supported. Therefore, ATMTransmitterFEP, ATMReceiverFEP, TCPTransmitterFEP, and TCPReceiverFEP further extend FEP by device-specific characteristics.

The MMDevices required are instantiated by MMDeviceFactory objects in the management service and bound to FEPs. For this purpose, a list (con_param_list) is generated containing all connections and connection parameters as previously described. In addition, when a binding takes place for a device in the list, the variable new_device is set to TRUE in order to indicate that a new device has been added to the list and to awaken sleeping processes.

After the required network connections have been set up, the associated FEPs are bound to each other via the operations connect and accept. The FEPs are now active and ready to pull or push

data. When the MMDevices are started, the contents of the buffers are exchanged between the devices.

With the MMDevices implemented thus far, the following data transfers are enabled:

- File source /MPEG encoder: push These devices are only able to provide data in the push mode.
- File sink / MPEG decoder: pull These devices are only able to request data in the pull mode.
- FEP / filter: push/pull These devices are able to support both the push and the pull mode.

As soon as MMDevice is started, it begins to scan the list of bound devices as an independent thread. If the search encounters a necessity to shift data, then a *push* will be executed. This is the case if, for example, a source MMDevice finds a device with the direction parameter set to *in* and the buffer access to *push*. Each bound device is processed in the same manner.

If a device is inactive, then it will be suspended with the operation *wait* and can be activated with *notify* if new devices have been added. The operation *unbind* is used to delete devices in the list of bound devices.

6.4 Buffer Management

An instance of class RingBuffer is associated with each device. The ring buffer is an efficient way to temporarily store data between two devices. Unfortunately, in most cases data cannot be transferred directly from the memory of one hardware device to the memory of the bound device, e.g. from a hardware audio/video encoder to a networking device. The main reasons are either limited memory capacity in these devices or the fact that such direct transfer mechanisms are not supported by standard operating systems. On the other hand, one of the main objectives of this thesis is to support different kinds of devices via standard APIs and achieve hardware and operating system independence as soon as possible. Nevertheless, communication systems which are, in principle, able to provide direct device-to-device communication from, for example, a DV camera to hard disk via Firewire [IEEE1394], [KuWe95] can also be handled with the proposed architecture. Implementation details of the buffer management can be found in section 8.4.3.

6.5 JMF Integration

Access to the JMF playback functionality is provided in different abstraction layers within Java. In the highest layer, the player can be started via a *URLStreamHandler* which specifies the location and the transport protocol by means of a URL (Unified Resource Locator). Although this is the simplest way to access the media player, this method provides less flexibility. To be able to deploy an own QoS-aware transport protocol and an own device control, the media player is accessed at a lower layer. Figure 6.6 shows the objects involved and their dependencies. The *OwnDataSource* and *OwnSourceStream* objects are built based on the *DataSource* and *SourceStream* class to be connected, for instance, with the *nativeATM_FEP*. The *OwnDataSource* object uses the JNI of the QoS-aware FEP device. Registration of the *nativeATM_FEP* is achieved through the *getStream* method from the *PullDataSource* class. A push model is also available. The *OwnDataSource* object is registered in the *Manager*. Using the return of the *getContentType* method, the *Manager* selects an appropriate *Player* and registers the *OwnDataSource* object within this *Player*.

To register the *ATMSourceStream* in the new *DataSource*, the *getStreams* method from the *PullDataSource* class is used. The method *getContentType* returns the content type of this data source. With this content type, the *Manager* can create the right *Player*.

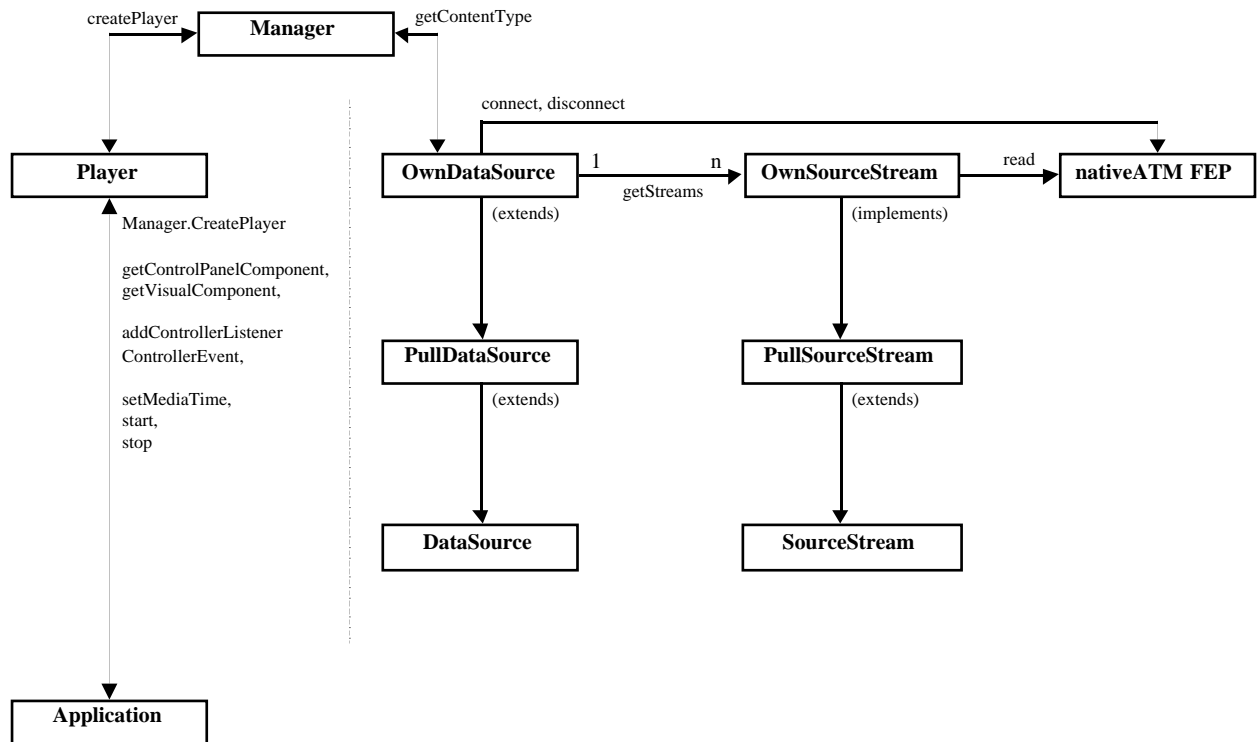


Figure 6.6: Integration of JMF Player Functionality

6.6 Summary

The three-layer configuration of the stream control and management architecture has been introduced: (1) transport layer, (2) device layer, (3) management layer. The object model has been specified by using UML as a notation. The communication model and buffer management as foundations for the architecture have been explained in detail. Finally, the feasibility of the integration of JMF-based devices has been discussed.

7 Management Service

This chapter describes the architectural concept of the management layer, its interaction with the device layer, and a prototypical implementation of the graphical user interface (GUI).

7.1 Architectural Concept of the Management Layer

In the initial state of the system, the servers running on the hosts provide factory objects for the different devices and flow endpoints (FEP). The factory objects are embedded in a HostManager object registered within the CORBA naming service. Factory objects and the HostManager objects provide a CORBA interface (Figure 7.1).

One HostManager object is instantiated in each end system. The HostManager objects list the factory objects available and provide management and monitoring operations for a host.

When a connection is to be established, the appropriate instances of MMDevice and FEP objects can be instantiated via the factories. Each instance is only responsible for a specific connection. The instances have CORBA interfaces to enable remote control and management.

The systems are bound to each other via CORBA interactions. After this set-up process, the systems can be remotely controlled via their specific commands.

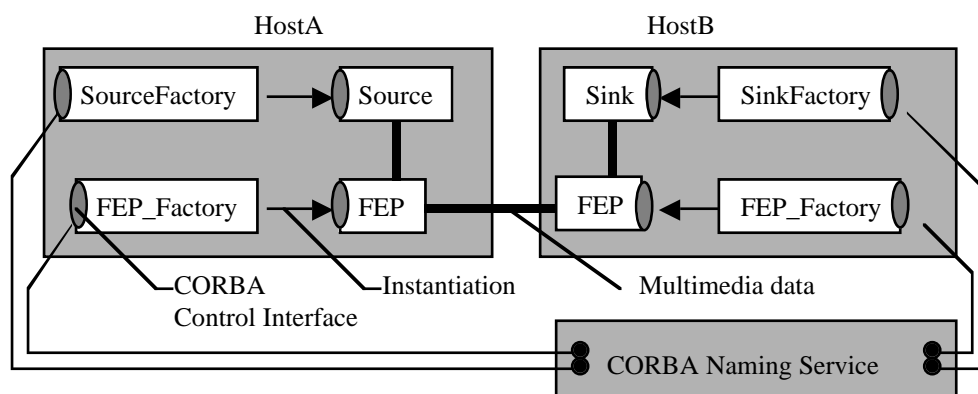


Figure 7.1: Main Elements of the Architecture at the CORBA Level

Source, Sink, and FEP, as shown in Figure 7.1, are placeholders for the different subclasses that can be placed in the system. Figure 7.2 depicts a scenario with different types of devices and flows. Please note that multiple virtual encoder devices can be instantiated, although only one physical encoder is available. The different instances of the encoder object represent the encoder in the context of one specific connection.

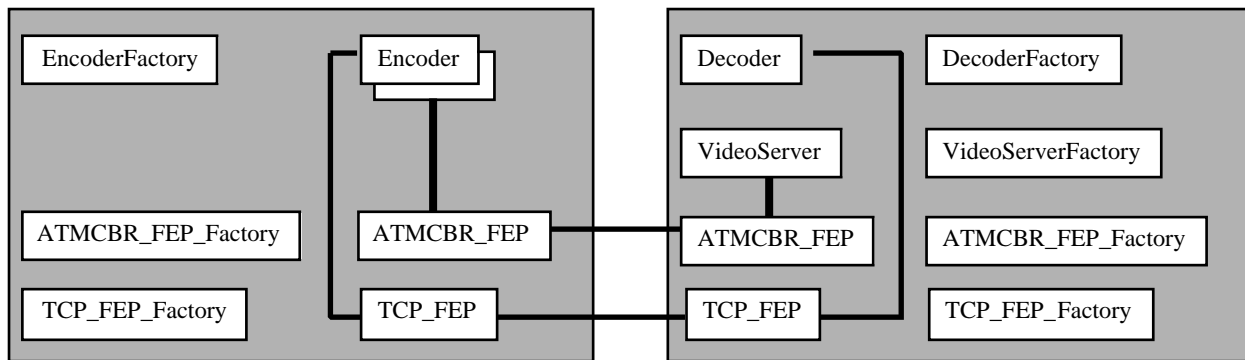


Figure 7.2: Multiple Connection with Different Devices and Streams

Each of the objects of the management layer (HostManager, FactoryObjects) and the corresponding management objects for device objects (MMDevices, FEPs) provide an interface for reading and writing of their properties. Properties allow further classification of objects (source, sink, device type, etc.) and the definition of capabilities, which can be set for each object instance. Capabilities are, for example, the bandwidth limit for an MPEG-2 encoder or the number of instances that can be generated from a factory.

For portability reasons, this approach uses Java as a programming language for the implementation of the different services. Because many of the APIs that come with encoder, decoder, and ATM network interface cards provide support for C/C++ only, some parts of the system have to be programmed in C/C++ and integrated in Java via the Java Native Interface (JNI). This leads to a layered architecture consisting of a high-level CORBA interface, which accesses a lower-level Java implementation that uses native code to gain access to the lower-level resources.

7.2 Interaction with the Device Layer

The implementation of the FEPs and MMDevices is mainly done with Java and C++ classes which are integrated via the JNI. These Java classes are wrapped with CORBA classes that provide remote access to the FEPs and MMDevices. CORBA factory classes are provided for the instantiation of new objects. Factory classes are embedded in HostManager objects registered within the CORBA naming service. Already instantiated objects are administered via the corresponding HostManager objects. That means that a client can get a list of already instantiated FEPs from the appropriate HostManager of the specific host and use the reference of an FEP to connect additional MMDevices. The HostManager object provides a higher-level grouping and management of CORBA objects.

Establishing New Connections and Bindings

The architecture proposed in this thesis enables the setup of a network of connections between FEPs and MMDevices. New elements can be connected to already instantiated elements or to newly created objects. The integration of new objects into the network has to follow certain rules. The GUI of the management application should only provide such alternatives to the user as are allowed by these rules. The rules are as follows:

- A source MMDevice can be connected to one or more FEPs or to another MMDevice.
- A sink MMDevice can be connected to only one FEP.
- A filter MMDevice is connected to one FEP to receive data from and can forward its data to several FEPs.
- An FEP can only receive data from one MMDevice.

- An FEP connected to a source MMDevice has exactly one network connection to another FEP of the same type.
- An FEP on the receiving side can forward its data to several MMDevices.

To summarize these rules, when considering the data flow from a source MMDevice to several destination MMDevices, the flow can be split up to be sent to more than one receiver. This splitting up can only be done within a host (0..n multiplicity for associations from source to destinations). Network connections between hosts have 1..1 multiplicity between two FEPs of the same type.

In accordance with these rules, the following interaction flow is provided to the user by the GUI of the management application. Available MMDevice means that an MMDevice is instantiated from a previous session and is able to satisfy the required QoS constraints with respect to hardware and software resources. This is also valid for FEPs.

1. Select a source – three alternatives are possible:
 - a) Instantiate a new MMDevice from the MMDeviceFactory and GOTO 2.
 - b) Use an already available MMDevice and GOTO 2.
 - c) Use an FEP that receives data from another FEP and GOTO 4.

Depending on the user selection, the program branches to the given step.
2. Select an FEP that is able to accept data from the MMDevice or shortcut to another MMDevice that is capable of receiving data:
 - a) Instantiate a new FEP from an FEP_Factory and GOTO 3.
 - b) Instantiate a new destination MMDevice and GOTO END.
 - c) Use an already available MMDevice (ActiveNode) and GOTO END.
3. Select an FEP that receives data over a network connection from another FEP:
 - a) Instantiate a new FEP from an FEP_Factory and GOTO 4.
4. Select a sink – two alternatives are possible:
 - a) Instantiate a new MMDevice from an MMDeviceFactory and GOTO END.
 - b) Use an already available MMDevice and GOTO END.

END.

To be able to perform the different steps given in the flow above, some properties of the various MMDevices and FEPs must be known by the management service. For example, in step 1, only source devices should be presented by the GUI for selection. In step 2, only FEPs that are able to accept data from an MMDevice should be selectable. These capabilities are stored in a property table in each MMDevice and FEP. This property table is a hash table that holds (name, value) pairs as strings. The table is filled during instantiation from a factory and binding between FEPs and MMDevices. When already instantiated objects are used, the table is utilized to retrieve this information.

7.3 Graphical User Interface

This chapter introduces the basic concept of the user interface of the management application. The following sequence of screen snapshots shows how an end-to-end connection can be set up. Please

note that the focus is not the design of the user interface itself, but to visualize the functionality of the interface.

In advance, by starting up an end and intermediate system, each HostManager is registered within the CORBA name service. Hosts are added by pressing the “add Hostmanager” button (cf. Figure 7.3).

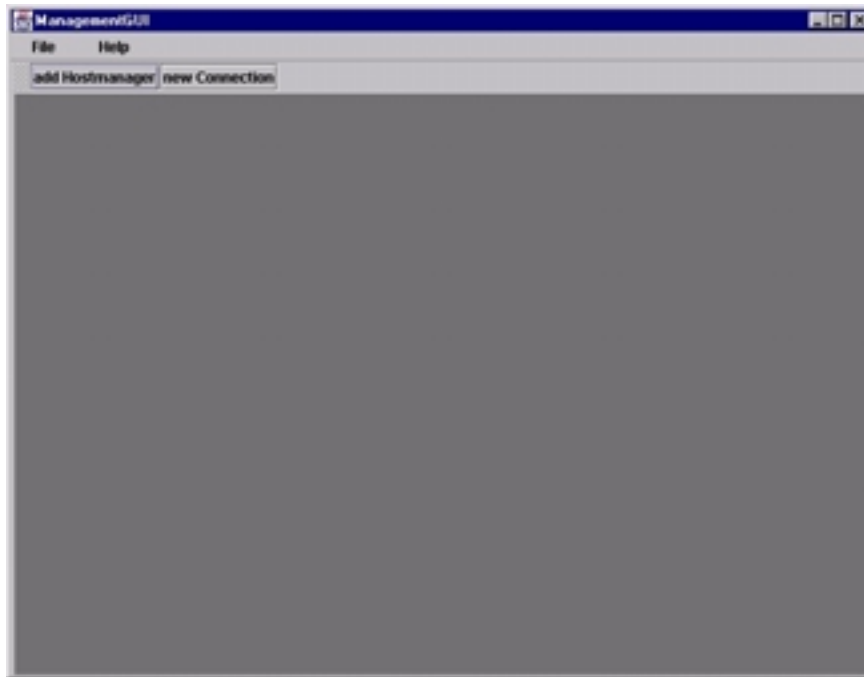


Figure 7.3: Basic Management GUI

Each HostManager selected then provides a list of available factory objects for devices and FEPs as depicted in Figure 7.4.



Figure 7.4: Example List of Available Factory Objects

Connections between end systems or intermediate systems are then set up by pressing the “New Connection” button (cf. Figure 7.3). A dialog then shows the component selected as illustrated in Figure 7.5, Figure 7.6, and Figure 7.7.



Figure 7.5: New Connection Dialog (1)



Figure 7.6: New Connection Dialog (2)



Figure 7.7: New Connection Dialog (3)

The final connection between two end systems is depicted in the figure below.



Figure 7.8: Connection Between Two End Systems

The dialog is controlled by the use of a finite-state machine. Each component displayed in a HostManager consists of a selection object which is responsible for handing over a selection to a dialog object. Each selection object holds information on the properties of an object and a reference

to the accompanying CORBA object. A list of selection objects is available at the end of a dialog, thus enabling the requested connections to be established. Afterwards, each component can be separately controlled and monitored by means of a pop-up menu.

7.4 Summary

The management layer architecture and its interaction with the device layer have been specified in detail. How a distributed system such as CORBA can be used to uncouple, on the one hand, and to interact, on the other, has been shown. The graphical user interface has been designed as a means of quick and easy access to effect interconnections: device-specific properties can be accessed via this GUI. Further services such as a QoS mapping and brokering component can easily be integrated by the same means.

8 Implementation

8.1 Software Structure

As already discussed, CORBA is the ideal operating system-independent distribution platform for realization of the above-defined architecture. At the same time, the use of Java enables the portability of components to other operating systems. This is of particular importance on the client side with end users deploying different operating systems.

However, an entire implementation of all of the components in Java cannot be done or is not meaningful in the following cases:

- For performance reasons, an implementation in native code (non-Java code) might be indispensable on the server.
- The Java platform has to be extended to the server and client to enable access to hardware and special APIs such as native ATM, hardware MPEG encoders and decoders, and video server APIs. Typically, these components can only be accessed via a C or C++ interface.

In the second case, the implementation with so-called native Java methods as wrapper classes is preferred in order that the required functionality at the Java level may be delivered. Later on, the implementation strategy outlined below will be applied [ERM99]:

As depicted in Figure 8.1, Native methods are used to access hardware-specific interfaces. In the case of the MPEG-2 encoder, native methods enable both the basic configuration of the encoder and starting and stopping of the encoding process. These methods are directly accessible in Java via the Java Native Interface (JNI). By embedding the object in a CORBA object, it becomes remotely available.

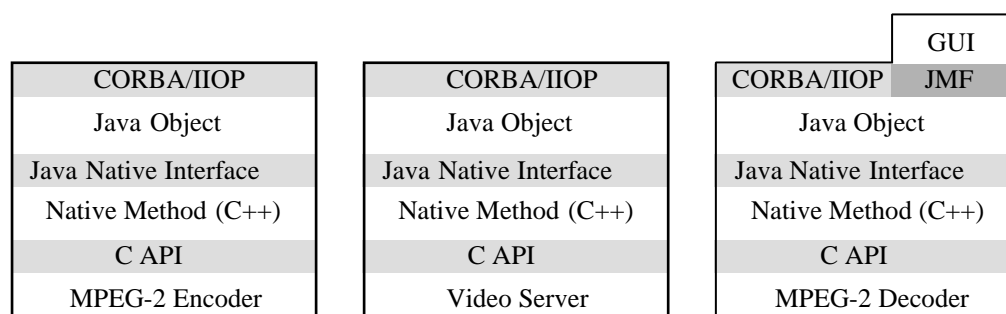


Figure 8.1: Use of Java/CORBA and Native Code

The video server is integrated in the same way, and an interface for configuration, control, and monitoring is provided. On the client side, the integration of multimedia playout devices is done through the JMF. As mentioned in section 3.3.3.1, JMF already supports a variety of media formats such as MPEG-1 and Quicktime. However, the integration of hardware-based MPEG-2 decoders is again done by the use of native code. Such a multimedia playout represents an MMDevice.

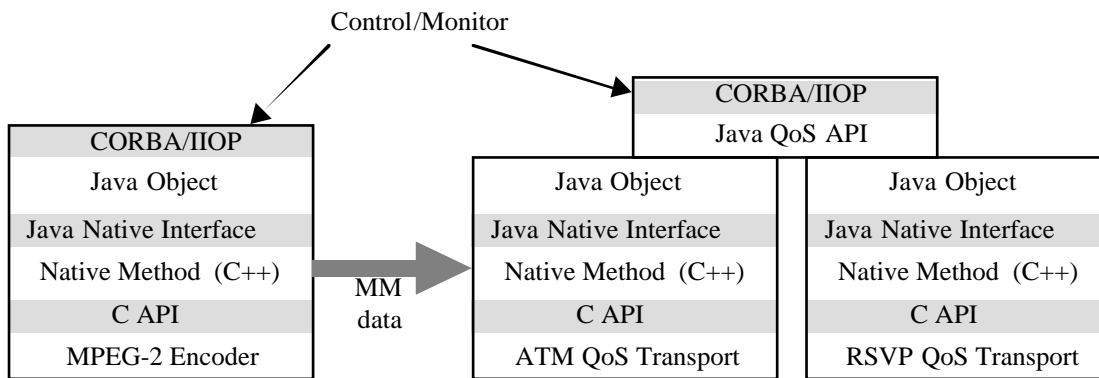


Figure 8.2: Java QoS API and Connection with Multimedia Devices

To provide support of native ATM and device-specific libraries, native code is integrated in the Java platform. For performance reasons, multimedia flows may be transferred directly within the native code – without propagation to the higher Java or CORBA level. In the current implementation, the flow is handled at the Java level. An interface to control the multimedia traffic from Java is realized in contrast to a Java-ATM API.

In the current implementation, the devices only support a single, unidirectional data flow at a time. This may change when the system is extended. A device that is used in a connection is classified to be a source or a sink object. A device can support both directions, e.g. a fileserver, but, during the set-up of a connection, the direction of the flow is selected. In the same manner, streams are simplified to support only a single, unidirectional data flow.

8.2 CORBA Interfaces

In order that different devices may be included in the system, interface inheritance was used at the CORBA level. A base interface is utilized for the joint operations of all of the devices. The interfaces of the devices which are to be included have to inherit from this basic interface “device”.

The complete IDL interface definition which has been used to wrap the device layer classes can be found in appendix 11.7. These CORBA interfaces correspond to the Java interfaces of the devices. The difference between the objects and their wrapping is that only the operations required for remote access are exposed. At the management level, the HostManager object provides a higher-level grouping and management of CORBA objects.

8.3 Java Implementations

As a rule, the CORBA interfaces are implemented in Java classes. Should it become necessary to access specific native libraries in order to, in turn, access the hardware, the Java native interface is used (see section 8.4.1). Nevertheless, the CORBA interface maps to a Java implementation class in the first step. Certain methods of the Java class may be implemented as native methods.

The CORBA interfaces of the different classes represent the management interface of the system. The actual data transmission is performed beneath this surface. When using native FEPs and devices, data may be transmitted direct at native level. If the FEPs and devices are pure Java objects, the transmission can be carried out in Java. At the current implementation state, data is always forwarded at Java level. A mixture of different kinds of objects necessitates data propagation from native code to Java and vice versa. Any combination has to be supported as set out in section 8.4.2.

Two major aspects have to be considered for direct data transmission:

- The implementation of the FEP must have a direct reference to the implementation of the local bound device.
- MMDevices and FEPs must provide methods for a direct hand-over of data to each other.

The first aspect is realized through a table which maps a CORBA object reference to a Java object which implements this interface. The same approach is used for the flow endpoint objects. When a new FEP is instantiated from a factory, a corresponding entry is made in the table. When a flow endpoint IOR is passed to an MMDevice object via the bind method, the device consults the table and acquires the implementation object and then registers its own implementation at the FEP, thereby ensuring that the implementation of the FEP knows the implementation object of the device and vice versa.

Two modes are taken into consideration for the direct data transmission between the implementation objects (cf. section 6.2):

- Push: data can be pushed from an originating device into an FEP for transmission and from a destination FEP to a device.
- Pull: an FEP can retrieve data from a device at the sender, or the receiving device may pull the data from the sink FEP.

Exactly which mode is used depends on whether or not a specific bandwidth is prescribed by the FEPs and MMDevices. If, for example, an encoder is configured for a specific bandwidth and the connection to the sink FEP supports this bandwidth, it is much more effective for the encoder to push the data to its associated FEP. However, should a FileSender be used as source and data is to be transmitted at the maximum data rate supported by the connection to the other FEP, then the FEP should pull the data from the FileSender.

To support push/pull models, both the FEPs and MMDevices support a push and a pull method for the forwarding of data at the Java level. The methods may be implemented using native methods. In addition, these methods may be called directly at native level via a similar mechanism.

8.4 Device Layer

8.4.1 JNI Interfaces

Generally, the Java programming language is used to implement the CORBA objects with the given interfaces. Hardware-specific native code is integrated via the Java Native Interface. Special JNI wrapper classes are utilized to provide an interface to the lower-level services in Java.

For example, when transmitting MPEG-2 material from an encoder to a decoder over ATM CBR, both decoder and encoder and the ATM network interface card are accessed through native code. Consequently, the decision is to not propagate any multimedia data to the Java level. Instead, the data is directly handed over from the encoder to the ATM library, and, at the decoder side, the data is directly forwarded from the ATM library to the decoder board. The JNI, thus, provides a Java interface for the control of the data flow in the native layer.

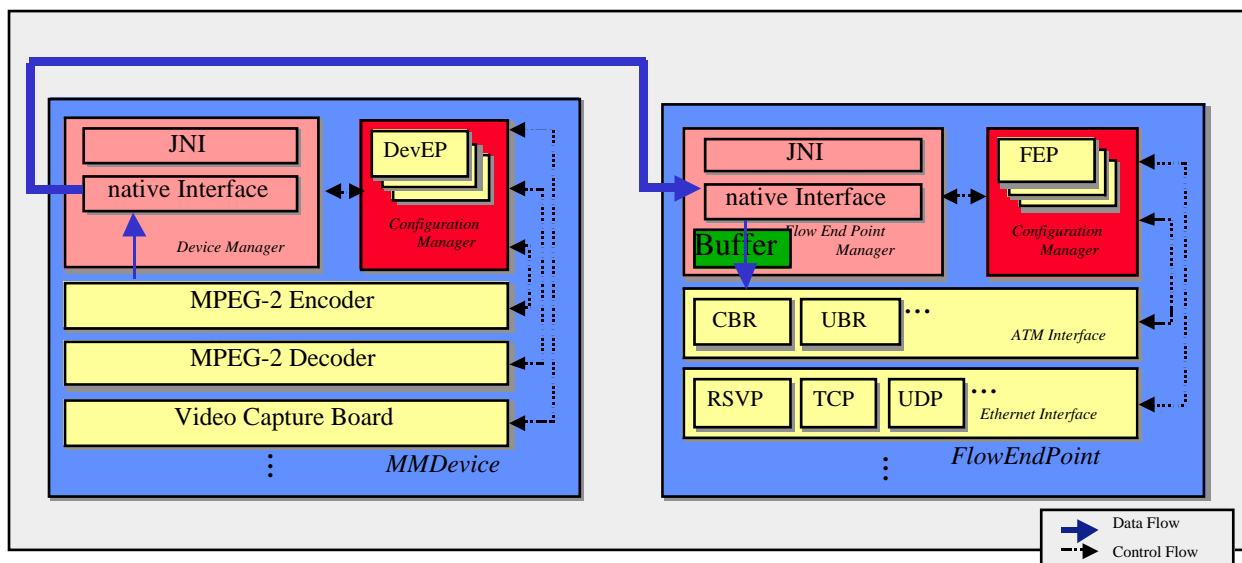


Figure 8.3: Integration of Devices with JNI

Figure 8.3 highlights further details of the implementation of the native devices. As clearly illustrated, the multimedia devices and stream endpoints have a very similar structure. The configuration manager knows the accessible hardware and software components at each node – like network interfaces or end devices – and shares these resources at the node between different virtual devices or flow endpoints.

Each configuration of a native device has two interfaces: one native interface and one Java interface. While the Java interface – implemented using the Java native interface definition – is also used for control purposes, the native or C interface is used solely for media transport. The advantage is that the media stream does not have to be shifted up to Java level when two native base devices are bound together, thus improving the performance of the system.

One major task of the JNI other than the provision of access to a hardware device is to manage how the multiple virtual devices at the CORBA level access the joint hardware and how the link between a multimedia device and a flow endpoint is achieved.

Figure 8.3 roughly illustrates the components within the JNI layer in one of the host systems. Each time a new flow endpoint is to be created, a new FEP record is inserted in the configuration manager. These records are globally available within the server process for all kinds of FEPs. The records include all the relevant information for a flow endpoint, such as the type of the FEP and a reference to the corresponding socket for the transmission of data. A JNI_FEP directly corresponds to an FEP at the CORBA level. When a new JNI_FEP is created, a handle (integer value) identifying the position of the FEP record within the configuration manager is propagated to the Java level.

As shown in Figure 8.4, when a bind operation to a multimedia device is performed on the same host, this handle is passed to the native code of the device (JNIDevice). After this call to bind, the FEP and the device on one host are “connected”, i.e. they know how to access each other at the native code level. If, for example, an encoder device is already bound to an ATM CBR_FEP when the start operation is performed on the device object, the device object receives the FEP record utilizing the handle from the bind call. Depending on the data in the FEP record, the native code then forwards the data to the corresponding network device. In this case, the socket data structure for the ATM CBR connection would be used.

This mechanism enables the interconnection of different devices with different FEPs. The JNI interfaces provide the operations required for the propagation of the calls in the CORBA interface (start, stop, pause, etc.).

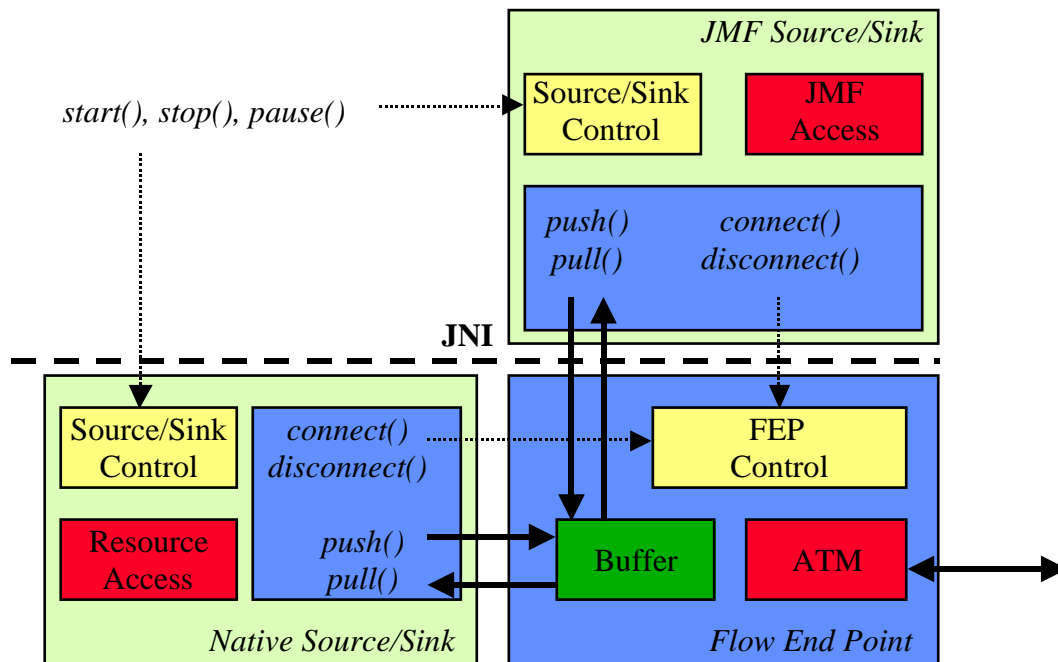


Figure 8.4: Base Device Binding

8.4.2 Interoperability of Java and Native Interfaces

For portability reasons, the majority of the interfaces should be programmed in Java. As pointed out above, there are some valid reasons for using native code. The four possibilities of connecting an MMDevice with an FEP are illustrated in Figure 8.5. Each native object has its representation on the Java side. Hence, the connections between Java and native objects are as simple as Java-Java connections. However, connections between two native objects are more complicated, as only their references have to be passed through the Java layer and the data is directly transmitted between the native objects.

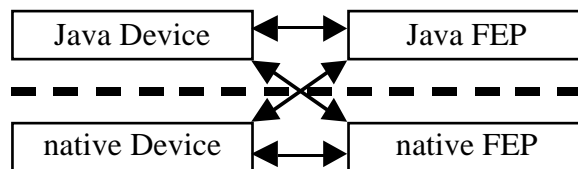


Figure 8.5: Interoperation of Java and Native Interfaces

8.4.3 Buffer Management

An instance of class RingBuffer is associated with each device. If a device wants to write to the buffer, buffer management first checks if enough memory is available to execute the operation. Second, if there is not enough memory available then the thread which wants to write will be suspended. Third, either other devices read data out of the buffer or the data becomes outdated because it has not been read in time, and buffer memory is freed for the waiting thread. Fourth, the device thread is finally able to write to the buffer.

Pointers are used to indicate the buffer area which is filled with data in use. The free buffer area can be computed by means of these pointers.

Each device with blocking access to a buffer has its own pointer, which is shifted by the number of read bytes after a successful read operation. After the rear pointer of all the available pointers has been moved, the global pointer indicating the amount of free and used buffer is shifted.

A write operation on a buffer can only be executed by one thread at a time. This prevents data from being overwritten by threads writing simultaneously. The buffer size is adjusted individually for each device according to its particular characteristics and QoS constraints. The smallest amount of data which can be read or written is one byte.

8.4.4 Device Binding

If objects want to exchange data, they have to be bound to each other. The binding takes place within the management layer by specifying which object should be bound to another object. It does not matter in which order objects are bound, i.e. whether object 1 is bound to object 2 or the other way around. Solely the flow direction has to be parameterized correctly. As described in section 6.2, the data flow direction is defined by the binding object, i.e. the device which initiates a data flow. Additionally, either the push or the pull mode of operation to be used has to be specified. Accordingly, at the end of the binding process, a push or pull thread will be instantiated. Threads are light-weight background processes, running independently of the main program.

As depicted in Figure 8.6, two kinds of binding are supported: internal and external binding. The easiest to understand and the conventional alternative is external binding. In this case, two devices such as an encoder and a network device are bound together. Internal binding is used, for example, when a source object has to access a file on a hard disk, i.e. the source object is to be bound to the file. This can be accomplished by using the push and pull threads.

This means that a source object desiring to read a file from a hard disk binds internally to itself by using the pull mode. Thus, the pull method of the source object will be overwritten by replacing the standard pull thread which would normally read-access the ring buffer of another bound object by reading the file.

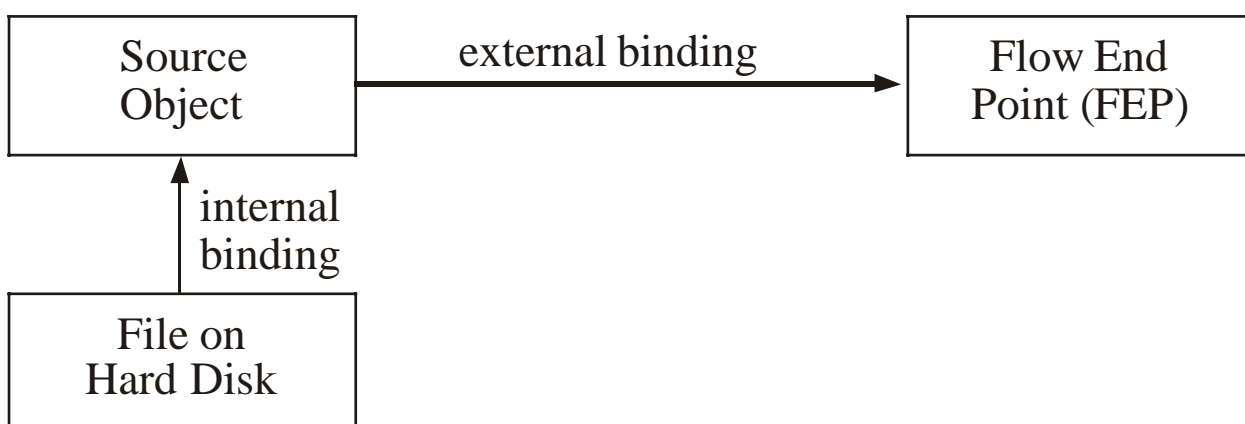


Figure 8.6: Internal and External Binding

Internal binding is required by all objects which do not just forward data from one ring buffer to another but also process data internally. As a rule, internal binding is used to describe the special behavior of an object. Internal binding is accomplished by the wrapper classes of the multimedia devices. A programmer who wants to add other multimedia devices does not have to worry about internal binding.

8.4.5 Push and Pull Threads

Push and pull threads are used to transfer data from one device to another (cf. Figure 8.7). Threads are used because they are light-weight processes which run in the background without blocking the main program. Additionally they can be set to inactive or put into sleep state.

There is a thread for each connection. The binding process creates the threads, which remain active until the data transmission has ended. The main tasks of the push and pull threads are to call their respective push and pull methods. Normally, these methods write data to or read data from the ring buffer. But, as already pointed out in the previous section, in the case of internal binding, these methods are overwritten.

BaseDevice

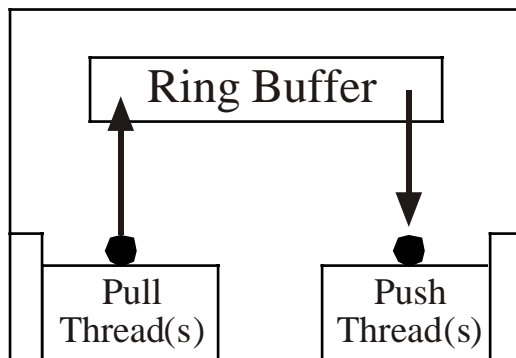


Figure 8.7: Base Device with Push and Pull Threads

8.5 Programming Tools

The following development tools were used to implement the architecture:

- Together 3.1, as an UML analysis and design tool.
- VCafe 3.0 and Borland JBuilder 3, for the rapid prototyping of the graphical user interface.
- OrbixWeb 3.2, a CORBA-compliant ORB implemented with Java.
- SDKs from Optibase, Fore, Sun, and Real Networks to access hardware and software encoders and decoders, and ATM communication systems.

8.6 Summary

Thanks to the object-oriented approach, the use of Java as the primary programming language, a CORBA-compliant distributed system, and object-oriented design and programming tools, it was possible to break the overall problem space down into smaller pieces in a straightforward manner. New communication and multimedia can easily be added without knowing the overall architecture. Hardware-oriented software for the access of devices such as hardware encoders, decoders or network interface cards can be consistently integrated into the framework using JNI. Furthermore, an in-depth explanation of how devices communicate with each other in the device layer has been given.

9 Experiments

This chapter gives an overview of the experimental environment for both the hardware and software in which the proof of concept for the architecture, design, and its implementation has been realized.

9.1 Application Area

As pointed out in section 1.2, the application area of this thesis is a distance learning application. The application includes the production, storage, retrieval, delivery, and presentation of content. Typically, content includes any kind of multimedia information needed to reach a learning objective. The most demanding pieces of information are the ones which require real-time end-to-end capabilities such as audio and video.

9.2 Experimental Environment

The experimental environment includes a prototypical system which is comprised of the main components of the system being used in a distance learning system deployed by my employer, DaimlerChrysler AG. The off-the-shelf hardware and software components used in the prototypical system have been enhanced by the software components developed within the scope of this thesis work.

9.2.1 Hardware

As depicted in Figure 9.1, the computers used in the experimental setup are standard off-the-shelf Intel Pentium II or Pentium III PCs. The ATM switch is a Fore ASX 200WG and the Ethernet switch is a Fore ES 3810, which provides the connection from the testbed to the Intranet. The PCs which are connected to the ATM switch use a Fore 155 Mbps ATM network interface card. The PCs which are connected to the Ethernet switch use standard 10 or 100 Mbps Ethernet network interfaces. The MPEG-1/-2 encoder uses Optibase's half D1 video encoding boards. In order to take advantage of hardware-supported decoding, Optibase's Videoplex and Videoplex Express boards are applied in the content manager and trainer PC. The media and video server is an SGI Origin 200. The Web-based training server is a server PC. A detailed list of the equipment used in the experimental environment is set out in appendix 11.8.

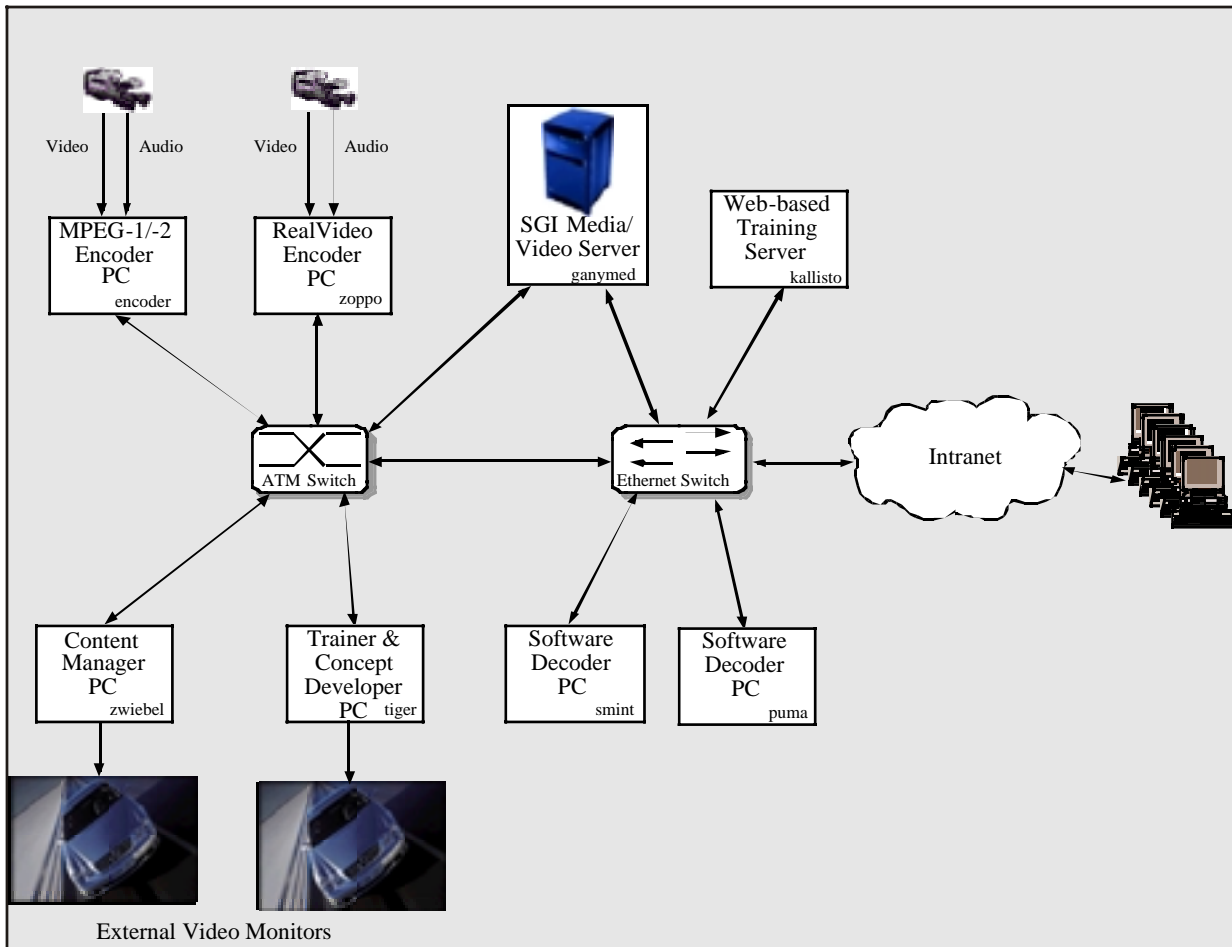


Figure 9.1: Experimental System

9.2.2 Software

All of the PCs employ Microsoft Windows NT 4.0 as the operating system. The SGI media server uses IRIX 6.5 as the operating system with a guaranteed-rate I/O subsystem. OrbixWeb from Iona Technologies is exploited as a Java-based implementation of CORBA.

The prototypical implementation consists of a management application that permits the remote configuration and management of the communication system and the multimedia devices with the corresponding server counterparts. Currently, the management application enables the setup of connections between each of the devices. As multimedia devices, FileSource, FileSender, MPEG-2 encoders and decoders and JMF decoders are supported. A splitter device has been employed as a simple active node. An ATM CBR connection and a best-effort TCP connection are implemented as communication links. As stated in section 7.3, after a set-up process, a GUI allows remote access to the devices.

A typical experimental configuration is shown in Figure 9.2. An MPEG encoder (encoder) is connected via ATM to a high-end client (zoppo) on one side and to a video server (ganymed) on the other side. The video server stores the video in its media repository, where it can be accessed by a content manager workstation (zwiebel). The high-end client (zoppo) is capable of decoding and displaying the streamed video and, at the same time, acts as an active node. In this example, the active node has two tasks: (1) transcoding of the video from high-quality MPEG-2 video format to a lower-quality Realvideo format and (2) transmitting the audio/video stream to a Realvideo server (kallisto) for storage and to a low-end client for decoding and display (smint).

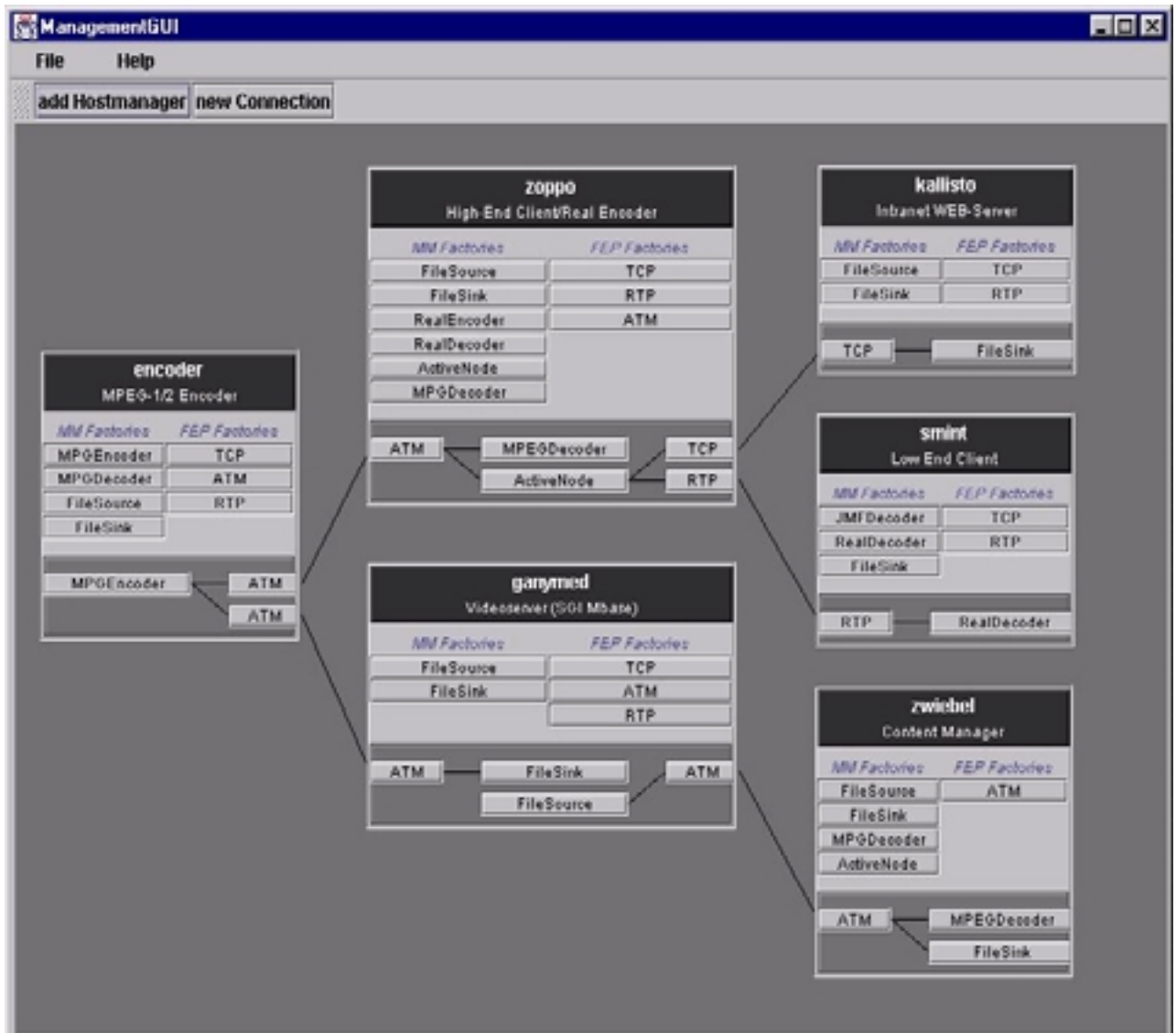


Figure 9.2: Experimental Configuration

9.3 Summary

The experimental results have shown that equipment from different vendors can easily be integrated in the experimental environment in a straightforward fashion. This is true both for multimedia devices that are hardware-based and for software-only solutions as well as for communication devices. Videos ranging from high to low quality could be streamed end-to-end without any problems:

- 4, 8, 12 Mbps encoded MPEG-2.
- 1 to 2.5 Mbps MPEG-1.
- 16 to 256 kbps low-quality video in RealG2, Quicktime, ASF and MPEG-4.

At the receivers, a streamed video could be stored and/or decoded in high quality as well as low quality. Both hardware and software-only solutions have been integrated in the experimental environment for decoding of videos. It has been shown that the concept of active nodes is viable and can be used for on-the-fly processing of streaming data.

Through the experimental setup, it has been proven that the overall concept and architecture of this thesis is feasible and can be used for a variety of application areas. The separation of the transport stream from its control and management enables quick-and-easy integration of new multimedia and communication devices.

10 Conclusions and Future Work

10.1 Conclusions

There is a huge demand for distributed multimedia applications which are typically interconnected via heterogeneous communication systems. The situation today in both commercial and research systems is characterized by the fact that distributed multimedia applications cannot use the functionalities of communication systems because they are buried in lower layers or at the right level of abstraction and, on the other hand, are not distributed by using state-of-the-art technology for designing and implementing distributed systems.

With respect to related work – in the context of CORBA – an architecture for the control and management of audio and video streams [MGRO97] has been specified. Initial implementations which support this standard have either not been completed and commercialized [IT98] or are built using a complex environment [MuSuSc99] and are, thus, too complex to adjust to the requirements of the applications. Therefore, my approach is a simplification of the CORBA A/V streaming standard proposal, with a light-weight architecture which is suitable for my specific environment and can be implemented using commercial off-the-shelf state-of-the-art components. Yet, the integration of or transition to an implementation whenever one becomes available and is suitable can still be done very easily.

For the support of ATM within Java, a Java ATM API was developed by the ATM Forum [ATMF98a]. As stated earlier in this thesis, my goal is to control the flow of multimedia data from the Java level. The flow itself is visible at the native or Java level. Due to the integration of multimedia devices which are already available in the Java platform, for example with the Java Media Framework (JMF), it was necessary to propagate the multimedia flow up to the Java level. In this thesis and in [ERR98], I have described how this can be accomplished.

I have designed a new light-weight end-to-end streaming architecture which is capable of using QoS features of end systems and communication systems. I have implemented a prototype of the stream control and management architecture by using commercial off-the-shelf components. New types of multimedia devices can be easily integrated in the architecture and the development framework. I have shown that it is feasible to separate stream management from the stream transportation. Subsequently, I have demonstrated how the lower-layer QoS capabilities of communication systems can be made available to application programmers at different levels of abstraction. My application environment was centered around eLearning applications. It was used for content production, content delivery, and content consumption. The environment allows the remote configuration, control, and management of multimedia and network devices and the streams between them. Devices which are not accessible from the Java platform are integrated using the Java Native Interface. The system was validated in a small lab environment as well as in a large network testbed which is used for the roll-out of network components at major sites of the DaimlerChrysler organization.

10.2 Thesis Contribution

The major contributions of this thesis are as follows:

- The overall design of an architectural framework to consistently manage multimedia devices and streams.
- From a developer's perspective, the framework allows rapid and flexible addition of new multimedia and communication devices to a system.

- From an application developer's point of view, a distributed multimedia application can now be handled like any other distributed application.
- The design of a layered system which separates the device layer from the management layer.
- The design of an application programming interface which is communication and operating system independent, but QoS aware.
- The design of a mapping mechanism which is able to provide the capabilities of devices to the application programmer's context at different levels of abstraction.
- The design of mechanisms which control media streams by applying the same concepts.
- The successful validation of the architecture through a distance-learning application by building an experimental prototype which helped to select appropriate mechanisms, identify new research issues, and identify limitations in existing systems.

10.3 Future Work

Usually, large companies deploy a global Intranet. These Intranets make use of a variety of communication technologies (Ethernet – shared/switched/fast), Token-Ring, FDDI, ATM, Frame Relay). The sites are interconnected and have a diversity of data rates ($n * 64$ kbps, 1.5, 2, 155 Mbps, 4 Gbps). For this reason, learners often experience a divergent data rate, which varies depending on the time of day, from their workplace to the central learning media server. So, future work will focus on a distributed system with central media servers and caching and replicating servers at remote locations, thus providing users with enhanced and easier-to-guarantee quality of service without a huge investment in an overall QoS-capable Intranet and Extranet. These remote servers can be managed using the same principles as those deployed in the architectural framework.

Further extensions will include the management of multimedia devices in local and remote classrooms. The intention is that trainers should not be bothered with technical issues such as setting up the client, server, and the network to the appropriate configuration with respect to the requested media. Often-used configurations can then be stored in usage profiles and automatically reconfigured the next time.

The abstraction and specification of QoS parameters at the user level and their mapping to the management layer is still an open research issue. However, a QoS management service which is responsible for QoS mapping and brokerage can easily be integrated in the architectural framework by applying the design principles used in my thesis. Another open question is which of the encoding parameters have to be handed over to the management level and how they should be presented.

With respect to the management application itself, the following open issues should be addressed in future work:

- If connections between devices are already configured and an operator accesses the system, the graphical user interface should automatically show any existing network of connections.
- A new component “NetManager” as already described in the IDL in appendix 11.7 is responsible for the management of a network of flows:
 - NetManager administers a list of HostManagers belonging to a network of flows.
 - Upon opening the GUI, a list of available NetManagers will be presented or the creation of a new instance requested.
 - NetManager will be registered in the CORBA name service.

- NetManager allows the operations on groups of devices such as start, stop or pause for all the multimedia devices of a network of connections.
- NetManager permits the monitoring and management of resources in order to solve conflicts.
- In order to support the functionalities of the NetManager, the HostManagers' grouping function has to be extended by QoS management and group operations.

Applying the architectural concepts to the upcoming area of active networks research is an interesting topic which is still unresolved. Through the simple implementation of a transcoding and splitter active node, I have shown that the architecture is also generally applicable for this area. However, further investigations will be required to bear this out.

11 Appendix

11.1 Video Compression

This part of the appendix describes the standardized video compression techniques used in the context of this thesis. As already mentioned, a detailed overview of compression techniques in general and on video compression is given in [EfSt98].

11.1.1 H.261

The H.261 method (also p*64 standard) was developed especially for video conferencing via ISDN (H.320 standard) and was standardized in 1990. It is a symmetric real-time compression method (same performance required for decoding and encoding). This standard defines two fixed resolutions for the transmitted images (CIF and QCIF, cf. Table 11.1). H.261 compression is characterized by low latency. And increasing the bandwidth results in improved image and sound quality. In 1995 the greatest video bandwidth supported was extended from 384 kbps (6 ISDN B channels) to 1.92 Mbps (30 ISDN B channels) and, if the appropriate software and hardware is available to support this feature, the transfer rate may now be selected within the range of 40 kbps to 1.92 Mbps as desired. However, the majority of the videoconferencing systems currently deployed are only able to support maximum transfer rates of 384 kbps. Videoconferencing systems in accordance with the H.320 standard generate a constant bit rate (CBR) in the network similar to the rate achieved by ISDN connections.

11.1.2 MPEG-1

MPEG-1 compression was developed for media using a low bandwidth (max. bit transfer rate 1.5 Mbps). In contrast to H.261, the resolution is not fixed; instead, a large range of parameters is defined. In order that overdimensioning of MPEG-1 implementations may be avoided, a so-called *constrained parameter set* (CPS) was introduced. A decoder or encoder fulfils these requirements, if it fulfils the parameters set out in Table 11.1 according to a *standard interchange format* (SIF). MPEG-1 enables extended features such as those available on a VCR (e.g. fast forward and rewind) to be applied to videos. Owing to the algorithm, the latency of MPEG-1 compression is higher than that of the H.261 method. For compression purposes, MPEG-1, as well as MPEG-2, which is described in the following section, define different types of images to be created from a video data stream.

- *Intraframe images (I frames)* compress solely the one image, similar to the JPEG algorithm. Any redundancies with respect to preceding or subsequent frames are not taken into account. Hence, such images represent the poorest maximum compression and serve as a reference for the other types of images defined.
- *Predictive images (P frames)* refer to a preceding I or P frame, thus enabling better compression than I frames due to the exploitation of time and space redundancies. P frames may also be utilized as a reference for other images. If a P frame is to be decoded, the preceding I frame and any and all of the P frames lying in-between have to be decoded.
- *Bi-directional images (B frames)* require information from the preceding and/or subsequent I or P frame. They enable the greatest possible compression of all of the types of images. However, they may not be used as a reference for other images.

The order of the images in a coded data stream is termed the *GOP (group of pictures)*. Selecting different GOP structures may impact the quality, the data rate required, and the latency of the coding.

Audio data is sampled at up to 48 kHz (stereo) and transmitted synchronously. Nowadays, MPEG-1 is deployed in video CD systems, CD-I systems, game consoles, and for audio-on-demand in the Internet (MP3, audio coding only). In accordance with the standard, MPEG-1 generates a variable bit rate (VBR) in the network, which is independent of the type of images transmitted. As the encoding thus becomes simpler, the commercially available devices marketed by the different manufacturers generate CBR data streams, which may lead to an unnecessarily high reservation of bandwidths within the network. Although MPEG-1 is an asymmetric method (encoding requires more computing performance than decoding), relatively reasonably priced real-time MPEG-1 encoders are the state of the art today. The table below shows an overview of the H.261 and MPEG-1 parameters.

Standard	Frame Formats	Data Transfer Rate	Comments
H.261	CIF: 352x288x15Hz QCIF: 176x288x15Hz	40 kbps to 1.92 Mbps (max. 30 ISDN B channels)	Used in connection with H.320 video conferencing systems
MPEG-1	SIF: 352x240x30Hz 352x288x25Hz	approx. 1.4 Mbps	Developed for media with a lower bandwidth (e.g. single-speed CD ROMs)

Table 11.1: H.261 and MPEG-1 Parameters

11.1.3 MPEG-2

MPEG-2 is an improved MPEG-1 compression standard for deployment in other applications such as digital video transmission via satellite or broadband net in television. Some key terms used in connection with this standard are *distance video* (remote classroom, distance learning, telemedicine, etc.) or *video on demand*. MPEG-2 extends the motion compensation of MPEG-1, thus enabling the effective coding of interlaced images (PAL or NTSC). Moreover, algorithms for scalable codings are defined. Scalable means that the quality of the decoder does not have to be the same as that of the encoder: the decoder needs only to evaluate parts of the video stream with respect to the quality and/or the resolution. The parameter ranges (maximum data transfer rate and the related maximum image size and frequency) are classified in *levels* and the coding methods in *profiles*. This classification is set out in and Table 11.3.

Profile	Comments
Simple	For software applications (software decoders), no bi-directional prediction of images (no B frames)
Main	Application for most of the decoder chips, not scalable, with bi-directional image prediction
Main+SNR	Same profile as main but scalable with respect to the quality (signal-to-noise scalability)
Main+spatial	Same profile as main but scalable with respect to the resolution (spatial scalability)
Next	All of the functionalities

Table 11.2: MPEG-2 Profiles

Level	Max. Image Size	Pixel/sec	Max. Bit Transfer Rate	Application
Low	352x288	3.05 m	4 Mbps	Corresponds to CIF (cf. MPEG-1)
Main	720x480	10.40 m	15 Mbps	CCIR 601, Studio TV
High 1440	1440x1152	47.00 m	60 Mbps	HDTV
High	1920x1080	62.70 m	80 Mbps	Production standard 240 m

Table 11.3: MPEG-2 Levels

Not all of the possible combinations of levels and profiles have been defined in the standard. Solely for the *main* level have all of the profiles been specified. In the following tables, the image qualities and the bandwidths required if the MPEG-2 standard is used are given for some resolutions. When calculating the video bandwidth required for transmission, the bandwidth required for the audio signals and a certain margin for overhead and additional data have to be added. The resulting values, which correspond to those specified in the data sheets provided by the manufacturers and are used in project planning, are entered in the third column of the table.

Dimension	Bandwidth After Compression	Max. Bandwidth for Transmission	Comments
352x480x24Hz (progressive)	2 Mbps	3 Mbps	Half-horizontal resolution similar to CCIR 601, almost NTSC quality, better quality than VHS (half D-1)
544x480x30Hz (interlaced)	4 Mbps	6 Mbps	PAL transmission quality
704x480x30Hz	6 Mbps	8 Mbps	Full CCIR 601 standard (studio standard, full D-1)

Table 11.4: Examples of Frame Formats and Bandwidths Using the MPEG-2 Standard

The latency of the MPEG-2 standard is similar to that of MPEG-1. Like MPEG-1, MPEG-2 includes synchronous audio coding and also generates a variable bit rate (VBR) in the network, which depends on the type of images to be represented (however, currently most MPEG-2 encoders are only able to generate constant bit rates). Furthermore, MPEG-1 stereo coding has been extended to multi-channel coding (3/2 stereo format) in the MPEG-2 standard. Both audio compression standards are compatible forward (MPEG-1 decoders are able to decode stereo signals sent in the MPEG-2 standard) and backward (MPEG-2 decoders are able to decode MPEG-1 audio signals). In addition, the MPEG-2 video compression standard is an asymmetrical process where encoding requires greater performance than decoding of an MPEG-2 data stream, thus resulting in enhanced hardware requirements and incurring higher costs for an MPEG-2 encoder.

The MPEG-2 data flow is hierarchical in structure, cf. Figure 11.1. In the encoder, so-called *elementary streams (ES)* are generated by compressing the video and audio signals. The data streams generated are packed in packets of a fixed or a variable length and, together with additional information such as time references and data transfer rate information, these packets form a so-called *packetized elementary stream (PES)*. Multiplexing of the different video and audio PESes generates a *system stream*. Two streams are differentiated: the *program stream* and the *transport stream*. Whereas a program stream is only able to transport a single MPEG-2 data stream and is, therefore, optimized for multimedia applications (e.g. for storage on a medium) and MPEG-1 compatibility, a transport stream is capable of transmitting several programs (video data streams), enables switching between programs and is optimized for transmission via a network. Furthermore, a program stream can also contain packetized, binary data from other applications, thus enabling, for instance, the transmission of arbitrary data in a logical data stream.

Despite the high computing performance required, the encoders available in the market today generally have MPEG-2 real-time coding capability. Differences may be found in the resolution achieved in coding the images, the vertical refresh rate, and the quality of the image transmitted at

MPEG-2 Data Streams

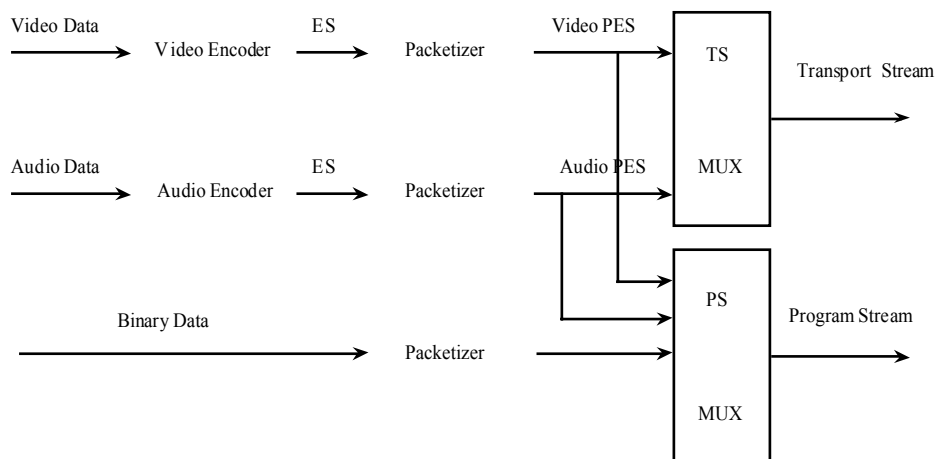


Figure 11.1: MPEG-2 Data Streams

the same required bandwidths. The best compression is achieved with bi-directional predictive images (B frames). Exactly how these frames are computed is not fixed in the standard. In particular, the size of the image area to be included in a prediction is not set. As some manufacturers deploy adaptive methods (*adaptive motion prediction, adaptive field/frame coding*), differences in the quality of the pictorial reproduction occur in spite of the fact that transmission was carried out at the same required bandwidths. Hence, testing of the various models available from the different manufacturers is a necessity, since the bandwidth required for transmission has a direct impact on the running costs.

Currently, the CCIR 601 standard seems to top the performance range for MPEG-2 real-time encoders (cf. Table 11.4, main profile & main level: MP@ML). For this specification and at a high compression rate, the delay due to coding lies within 200-400 ms. As low a value as possible has to be achieved through the selection of a suitable coding schema (GOP) in pilot testing. Encoders for

half-horizontal CCIR 601 resolution lie in the mid-performance range. Moreover, the MPEG-2 encoders and decoders made by most leading manufacturers are backward compatible and permit changing to the MPEG-1 compression standard.

However, the quality requirements set within this thesis project and, in particular, the planned reproduction of video data on large screens call for coding and data transfer by means of the MPEG-2 standard. Since MPEG-2 was developed especially for TV transmission technology, it has, for example, adapted the frame compression algorithm to interlaced scanning (as with PAL and NTSC). The best quality pictorial representation is currently offered by CCIR 601. Yet, the maximum data transfer rate required – 8 Mbps – has to be taken into account when planning the transmission technology.

Furthermore, deployment of the transport stream with its benefit of being able to transmit more than one program will enable a variety of innovative application scenarios in the future. In addition, improved transmission quality will be achieved through the implementation and utilization of higher profiles and levels (HDTV). All in all, MPEG-2 provides the greatest potential for future developments.

11.2 ATM Classes of Service

Table 11.5 shows the different classes of services specified by ITU-T and the ATM Forum.

ITU-T	DBR	SBR		ABR	--	ABT
ATM Forum	CBR	rt-VBR	nrt-VBR	ABR	UBR	--
Cell loss bound (CLP=0)	–	–	–	–	--	–
Cell loss bound (CLP=1)	– same as for CLP=0	(--)	(--)	--	--	– same as for CLP=0
CDV bound	–	(ATMF)	--	--	--	–
Cell delay bound	(ATMF)	(ATMF)	--	--	--	--
Traffic parameters	PCR, CDVT	PCR, CDVT, SCR, BT	PCR, CDVT, SCR, BT	PCR, CDVT, MCR	PCR, CDVT	PCR, CDVT, SCR, BT

Table 11.5: ATM Classes of Service

Key:

DBR: deterministic bit rate

CBR: constant bit rate

SBR:	statistical bit rate
VBR:	variable bit rate
ABR:	available bit rate
UBR:	unspecified bit rate
(n)rt:	(non) real-time
ABT:	ATM block transfer

11.3 Internet RFCs

This appendix cites the Internet RFCs which are relevant for this thesis. More information on the various working groups of the Internet Engineering Task Force (IETF) can be found at www.ietf.org.

IntServ RFCs include the following:

- The Use of RSVP with IETF Integrated Services (RFC 2210).
- Integrated Services Management Information Base using SMIPv2 (RFC 2213).
- Integrated Services Management Information Base Guaranteed Service Extensions using SMIPv2 (RFC 2214).
- General Characterization Parameters for Integrated Service Network Elements (RFC 2215).
- Network Element Service Specification Template (RFC 2216).
- Specification of the Controlled-Load Network Element Service (RFC 2211).
- Specification of Guaranteed Quality of Service (RFC 2212).

See the IETF IntServ page <http://www.ietf.org/html.charters/intserv-charter.html>.

Current Status of RSVP:

- The RSVP functional specification is described in the Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification (RFC 2205).
- RSVP Management Information Base using SMIPv2 (RFC 2206).
- RSVP Extensions for IPsec Data Flows (RFC 2207).
- Resource ReSerVation Protocol (RSVP) Version 1 Applicability Statement – Some Guidelines on Deployment (RFC 2208).
- Resource ReSerVation Protocol (RSVP) – Version 1 Message Processing Rules (RFC 2209).

A number of IETF drafts have been published. For further information, see the IETF RSVP page at <http://www.ietf.org/html.charters/rsvp-charter.html>.

DiffServ RFCs include the following:

- Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers (RFC 2474).
- An Architecture for Differentiated Services (RFC 2475).
- An Expedited Forwarding PHB (RFC 2598).
- Assured Forwarding PHB Group (RFC 2597).
- Per Hop Behavior Identification Codes (RFC 2836).

For further information, see the IETF Diffserv page at <http://www.ietf.org/html.charters/diffserv-charter.html>.

Policy RFC drafts:

- Policy Framework LDAP Core Schema.
- Policy Core Information Model - Version 1 Specification.
- QoS Policy Schema.
- Policy Framework QoS Information Model.
- Information Model for Describing Network Device QoS Mechanisms.
- Policy Terminology.

As yet, no RFCs have been published. For further information, see the IETF Policy page at <http://www.ietf.org/html.charters/policy-charter.html>.

Examples of Internet drafts produced by the RSVP Admission Policy (RAP) IETF working group are “A Framework for Policy-based Admission Control”, “The COPS (Common Open Policy Service) Protocol”, “RSVP Extensions for Policy Control”, and “COPS Usage for RSVP”. See the IETF RAP work group's page <http://www.ietf.org/html.charters/rap-charter.html>.

11.4 Multicasting

In a virtual classroom application, for example, it is often desirable to transmit information to a group of recipients via a single transmission by the source – in contrast to unicast and broadcast as described in this section. Whereas multicasting is a one-to-many transmission, unicasting is a point-to-point transmission requiring the source to send an individual copy of a message to each requester, and broadcasting is a one-to-all transmission where the source sends one copy of the message to all of the nodes – whether they wish to receive it or not.

From the technological perspective, there are several ways of distributing information over IP networks depending on the application and the traffic types. But in principle, there are three strategies:

Unicast is the most common type of link in the Internet. Unicast connections can be established anywhere in the Internet. Should a large number of receivers be interested in the same information, they all have to establish their own link to the information provider. This is extremely resource intensive and greatly limits scalability. Limiting factors are the processing power required to handle so many connections simultaneously and the capacity the network provides. A replication server can remedy these limitations.

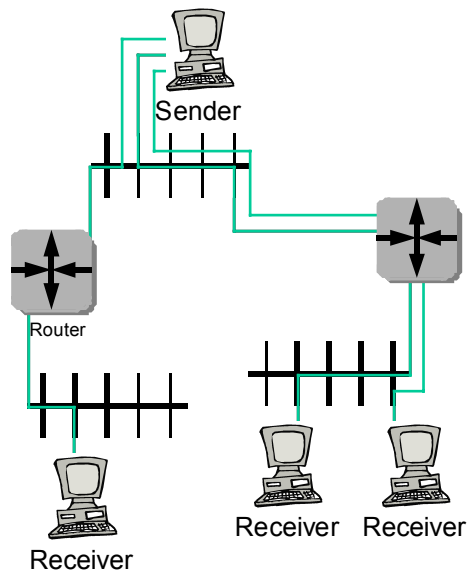


Figure 11.2: Unicast Transmission

Multicast is a much more elegant solution and conserves network resources. The information is only distributed once in the part of the network in which users wish to receive the information. In IP networks, there is a special address space for this kind of transport type. Multicast has to be implemented in all of the nodes along the network paths – and this is its major drawback. Most nodes in the Internet are not capable of deploying this technology. As a result, strategies have to be developed for the use of multicast in special Intranet islands. Although there are, in principle, no scalability problems with multicast, in reality, there is a need for higher layer protocols to ensure packet order and avoid packet loss, which – by themselves – may cause scalability problems. A high-quality survey of multicast protocols is set out in [Obra98].

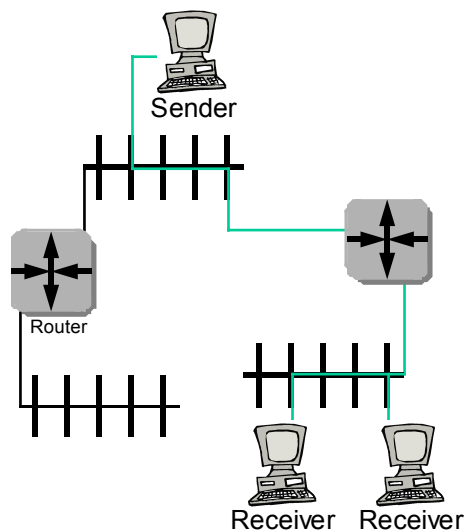


Figure 11.3: Multicast Transmission

Broadcast is the simultaneous transmission of the same information to all of the receivers within range, no matter whether they are interested in the information sent or not. In the Internet, broadcast often appears in connection with router queries. Since broadcast means flooding the whole network, network providers are afraid of the immense capacity demand a broadcast packet could cause – especially if the packet should run into a loop, which is quite likely. Most providers permit only one-hop broadcasts, substantially reducing the effectiveness of broadcasting in an Intranet. In our

context, we found broadcast in conjunction with typical broadcast mediums such as the satellite system and also terrestrial broadcast systems conceivable.

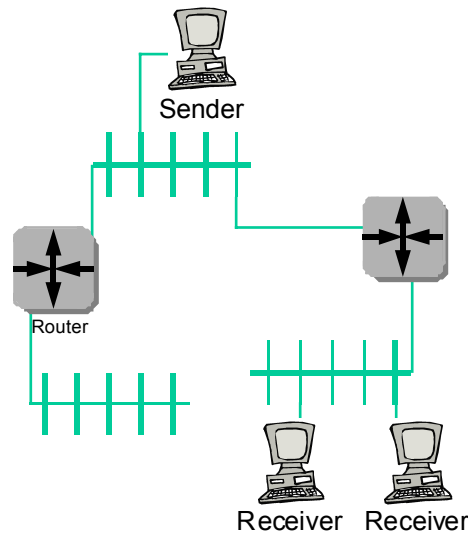


Figure 11.4: Broadcast Transmission

IP Multicast

IP multicast is an extension of the standard IP network-level protocol. RFC 1112, “Host Extensions for IP Multicasting”, authored by Steve Deering in 1989, laid the groundwork for IP multicasting. The RFC describes IP multicasting as: “the transmission of an IP datagram to a ‘host group’, a set of zero or more hosts identified by a single IP destination address. A multicast datagram is delivered to all members of its destination host group with the same ‘best-efforts’ reliability as regular unicast IP datagrams. The membership of a host group is dynamic; that is, hosts may join and leave groups at any time. There is no restriction on the location or number of members in a host group. A host may be a member of more than one group at a time.”

11.5 Microsoft Windows Socket API Version 2 Flowspec

Flowspecs describe the traffic characteristics of a proposed unidirectional flow through the network. An application can associate a pair of flowspecs with a socket via the `SIO_SET_QOS/SIO_SET_GROUP_QOS` command at connection set-up time using `WSAConnect()` or at any other time using `WSAIoctl()`. After a flow has been established, the application is notified of any changes in the available service level via `FD_QOS/FD_GROUP_QOS`.

The flowspecs proposed for WinSock 2 divide QoS characteristics into the following general areas:

Source traffic description – The manner in which the application's traffic is injected into the network. This includes specifications for the token rate, the token bucket size, and the peak bandwidth. Note that although the bandwidth requirement is expressed in terms of a token rate, this does not mean that service provider must actually implement token buckets. Any traffic management scheme that yields equivalent behavior is permitted.

Latency – Upper limits on the amount of delay and delay variation that are acceptable.

Level of service guarantee – Whether or not an absolute guarantee is required as opposed to best effort. Note that the connection attempt is expected to fail for providers who have no feasible way of providing the level of service requested.

Provider-specific parameters – The flowspec itself can be extended in ways that are particular to specific providers.

The WinSock 2 QoS structure is defined through a combination of the qos.h and winsock2.h header files. The relevant definitions are summarized below.

```
typedef struct_WSABUF {  
    u_long    len; /* the length of the buffer */  
  
    char FAR * buf; /* the pointer to the buffer */  
  
} WSABUF, FAR * LPWSABUF;
```

```
typedef uint32  SERVICETYPE;
```

```
typedef struct _flowspec  
{  
  
    uint32    TokenRate; /* In Bytes/sec */  
  
    uint32    TokenBucketSize; /* In Bytes */  
  
    uint32    PeakBandwidth; /* In Bytes/sec */  
  
    uint32    Latency; /* In microsec. */  
  
    uint32    DelayVariation; /* In microsec. */  
  
    SERVICETYPE ServiceType;  
  
    uint32    MaxSduSize; /* In Bytes */  
  
    uint32    MinimumPolicedSize; /* In Bytes */  
  
} FLOWSPEC, *PFLOWSPEC, FAR * LPFLOWSPEC;
```

```
typedef struct _QualityOfService  
{  
  
    FLOWSPEC SendingFlowspec; /* the flow spec for data sending */  
  
    FLOWSPEC ReceivingFlowspec; /* the flow spec for data receiving */  
  
    WSABUF ProviderSpecific; /* additional provider specific stuff */  
  
} QOS, FAR * LPQOS;
```

Definitions:

TokenRate/TokenBucketSize

A *Token bucket model* is used to specify the rate at which permission to send traffic (or credits) accrues.

The concept of the token bucket is a bucket which has a maximum volume (token bucket size) and continuously fills at a certain rate (token rate). If the bucket contains sufficient credit, the application may send data; if data is sent, the available credit is reduced by that amount. If sufficient credits are not available, the application must wait or discard the extra traffic.

If an application has been sending at a low rate for a period, it may send a large single burst of data until it runs out of credit. Having done so, it must limit itself to sending at *TokenRate* until its data burst is exhausted.

In video applications, the *TokenRate* is typically the average bit rate peak to peak, and the *TokenBucketSize* is the largest typical frame size. In constant rate applications, the *TokenRate* is equal to the *PeakBandwidth*, and the *TokenBucketSize* is chosen to accommodate small variations.

PeakBandwidth

This parameter limits how fast packets may be sent from the application back-to-back.

Latency

Latency is the maximum acceptable delay between transmission of a bit by the sender and its receipt by the intended receiver(s).

DelayVariation

This field is the difference between the maximum and minimum possible delay (jitter) that a packet will experience.

ServiceType

This is the level of service being negotiated for. The values permitted for a level of service are given below.

SERVICETYPE_NOTRAFFIC is used in either sending or receiving flowspec and indicates that there will be no traffic in this direction. On duplex-capable media, this command signals underlying software to set up unidirectional connections only.

SERVICETYPE_BESTEFFORT indicates that the service provider will, at the least, take the flowspec as a guideline and make reasonable efforts to maintain the level of service requested, however without making any guarantees whatsoever.

SERVICETYPE_CONTROLLEDLOAD indicates that end-to-end behavior provided to an application by a series of network elements closely approximates the behavior visible to applications receiving best-effort service under unloaded conditions from the same series of network elements. Thus, applications using this service may assume that: (1) a very high percentage of transmitted packets will be successfully delivered by the network to the receiving end nodes (The packet loss rate will closely approximate the basic packet error rate of the transmission medium.), and (2) the transit delay experienced by a very high percentage of the delivered packets will not greatly exceed the minimum transit delay experienced by any successfully delivered packet at the speed of light.

SERVICETYPE_GUARANTEED indicates that the service provider has implemented a queuing algorithm which isolates the flow from the effects of other flows as much as possible and guarantees the flow the ability to propagate data at the *TokenRate* for the duration of the connection. If the sender should send faster than that rate, the network may delay or discard the excess traffic. If the sender does not exceed *TokenRate* over time, then latency is also guaranteed. This service

type is designed for applications which require a precisely known quality of service but would not benefit from better service: for example, real-time control systems.

SERVICETYPE_NETWORK_UNAVAILABLE in either a sending or receiving flowspec, this may be used by a service provider to indicate a loss of service in the corresponding direction.

SERVICETYPE_GENERAL_INFORMATION indicates that all service types are supported for this traffic flow.

SERVICETYPE_NOCHANGE in either a sending or receiving flowspec, this requests that the QoS in the corresponding direction remain unchanged. This may be used when requesting a QoS change in one direction only or when requesting a change solely in the *ProviderSpecific* part of a QoS specification and not in the *SendingFlowspec* or the *ReceivingFlowspec*.

SERVICE_IMMEDIATE_TRAFFIC_CONTROL in either a sending or receiving flowspec, this may be combined using bit-wise OR with one of the other defined *ServiceType* values to request the service provider to activate traffic control coincident with provision of the flowspec.

MaxSduSize

This gives the maximum packet size, in bytes, that is permitted or used in the traffic flow.

MinimumPolicedSize

This parameter specifies the minimum packet size that will be given the level of service requested.

Via the `WSAGetQOSByName()` function, an application can obtain a template with the appropriate QoS structure for well-known media streams such as H.323, G.711, etc.

As described in [WSG96], the ATM-specific QoS extensions of Winsock 2 allow the definition of the data traffic descriptor with forward and backward peak cell rate (PCR), sustainable cell rate (SCR), and the maximum burst size (MBS). The `ATM_QOS_CLASS_IE` structure is used to signal the desired quality of service (classes 0, 1, 2, 3 or 4).

If the application uses the generic `FLOWSPEC` structure of Winsock 2, the ATM service provider has to map it to the appropriate Q.2931 information elements.

11.6 Object Model in Detail

The following series of figures shows the object model which was introduced in section 6.3 in more detail.

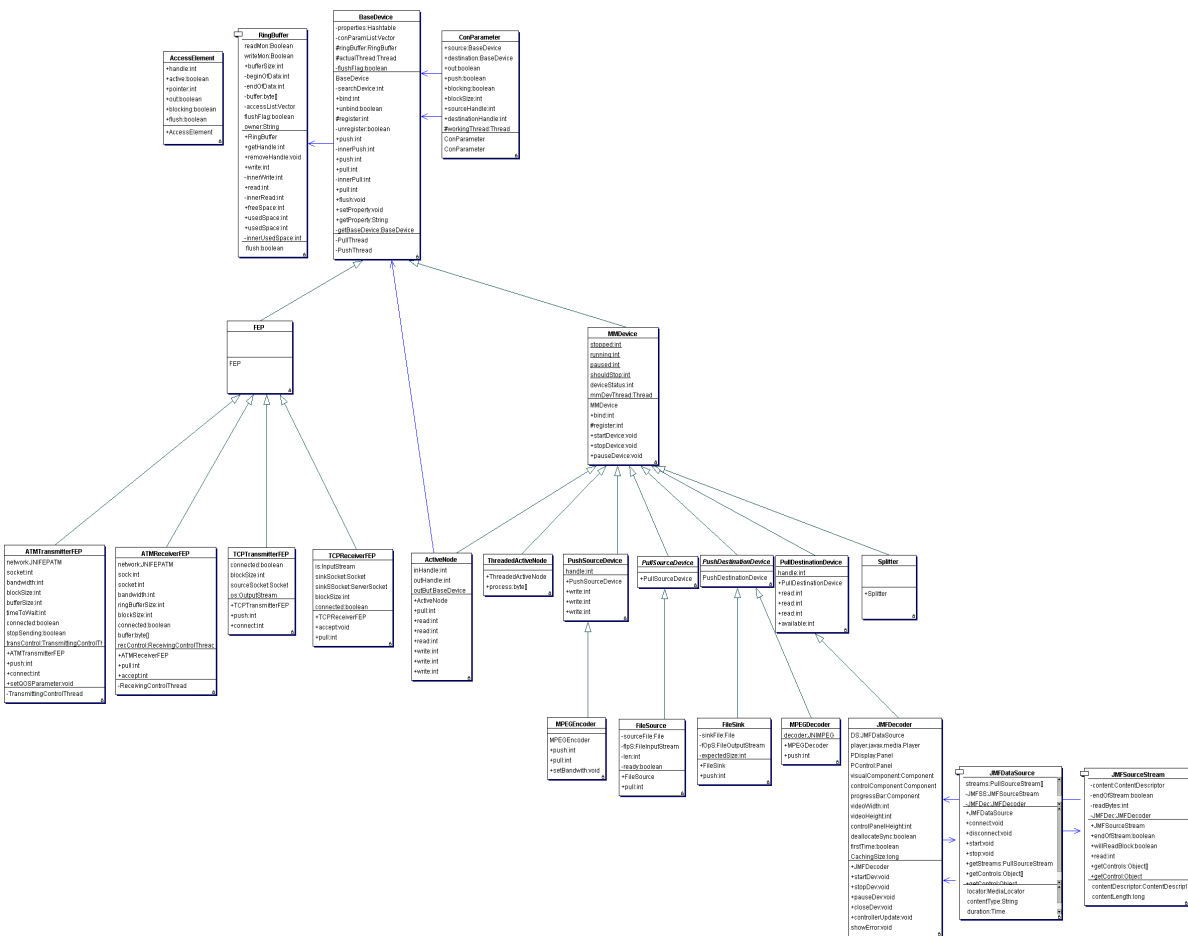


Figure 11.5 Object Model

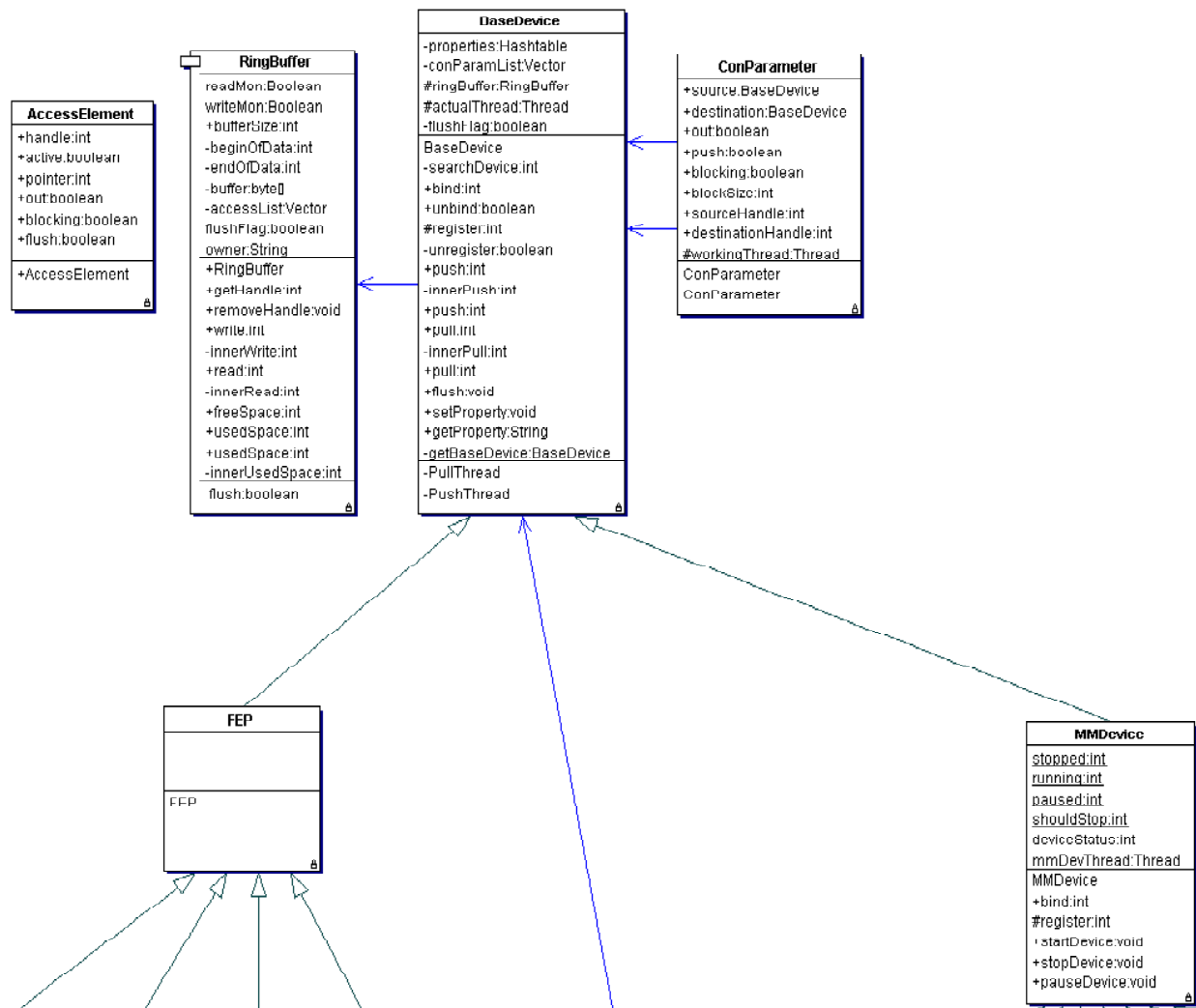


Figure 11.6: Object Model – Base Device, Flow Endpoint and Multimedia Device

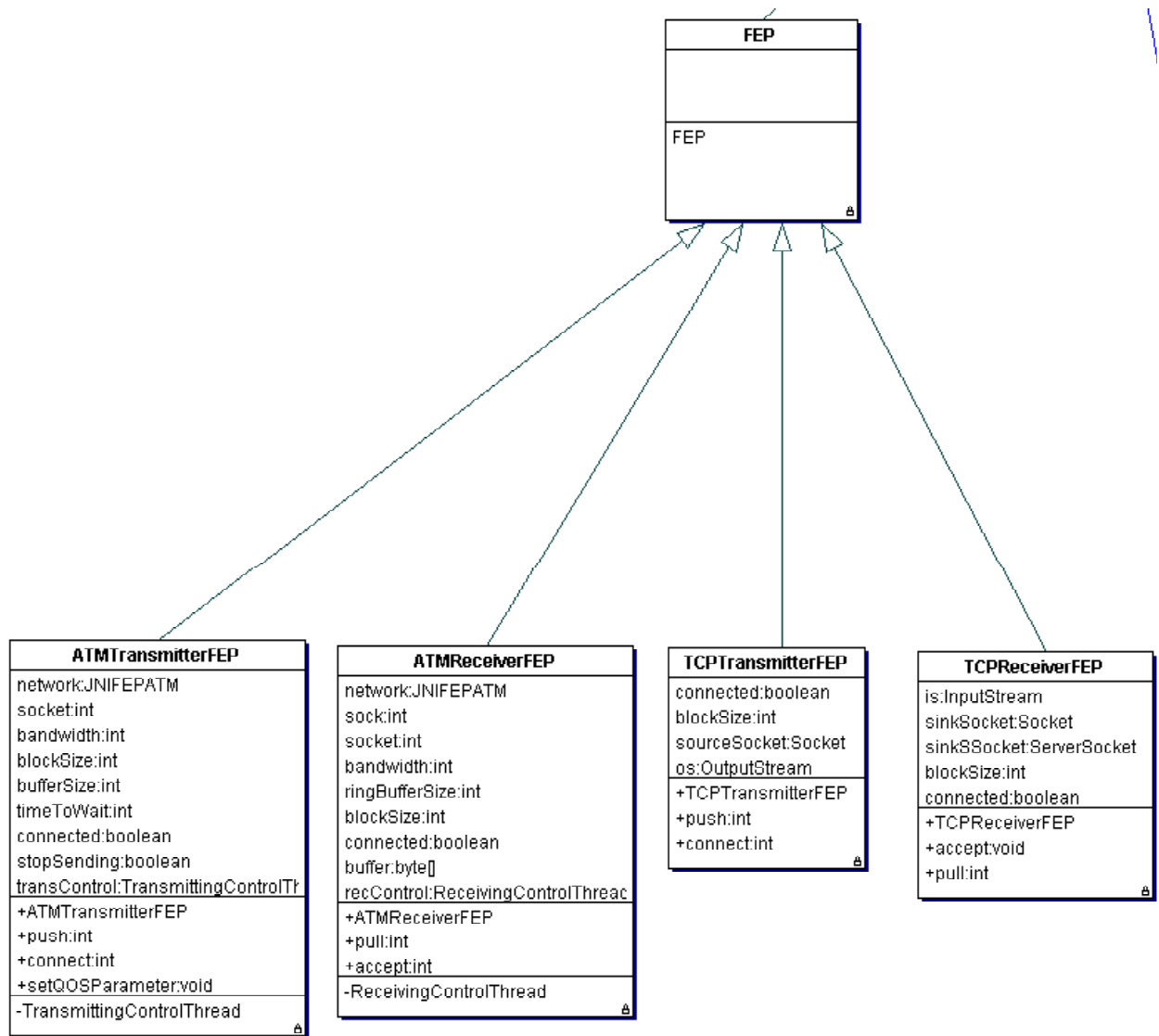


Figure 11.7: Object Model – Flow Endpoints



Figure 11.8: Object Model – Multimedia Devices

11.7 IDL Interface Definition

This section lists the IDL specification at the boundary to the management service.

```
enum DeviceOperation {start, stop, pause};
```

```
enum VideoFormat {MPEG1, HalfD1, FullD1};
```

```
enum FlowDirection {Source, Sink};
```

```

interface FEP;
interface MessageReceiver;
interface BaseDevice;
interface DeviceFactory;

struct ConParameter {
    BaseDevice source, destination;
    boolean isOut, push, blocking;
    long blockSize;
};

typedef sequence<ConParameter> ConParameterList;

interface PropertyObject {
    void setProperty(in string name, in string value);
    string getProperty(in string name);
    void registerMsgRcv(in MessageReceiver mr);
};

interface BaseDevice : PropertyObject {
    ConParameterList getConParameterList();
    void flush(in boolean f);
    long bind(in BaseDevice vd, in boolean isOut, in boolean push, in boolean blocking, in long blockSize);
    boolean unbind(in BaseDevice s,in BaseDevice d);
};

typedef sequence<BaseDevice> DeviceList;
typedef sequence<DeviceFactory> FactoryList;
interface HostManager : PropertyObject {
    void addDevice(in BaseDevice d);
    DeviceList getDeviceList(in string deviceType);
    DeviceList getConnectedSinkDevices(in BaseDevice source);
    DeviceList getConnectedSourceDevices(in BaseDevice sink);
    FactoryList getFactoryList();
};

typedef sequence<HostManager> HostManagerList;
interface NetManager : PropertyObject {
    void addHostManager(in HostManager hm);
    HostManagerList getHostManagers();
    void operation(in DeviceOperation op);
};

```

```
interface MMDevice : BaseDevice {  
    void operation(in DeviceOperation op);  
};
```

```
interface MPEGEncoder : MMDevice {  
    void setFormat(in VideoFormat format);  
    void setBitrate(in double kbps);  
};
```

```
interface MPEGDecoder : MMDevice {  
};
```

```
interface FileSource : MMDevice {  
//    void setFilename(in string n);  
};
```

```
interface FileSink : MMDevice {  
//    void setFilename(in string n);  
};
```

```
interface ActiveNode : MMDevice {  
};
```

```
interface FEP : BaseDevice {  
};
```

```
interface ATMTransmitterFEP : FEP {  
    long connect(in string hostName, in long selector);  
};
```

```
interface ATMReceiverFEP : FEP {  
    long accept(in long selector);  
};
```

```
interface TCPReceiverFEP : FEP {  
    void accept(in long portnr);  
};
```

```
interface TCPTransmitterFEP : FEP {  
    long connect(in string hostName, in long portnr);  
};
```

```
interface DeviceFactory : PropertyObject{
};

interface MPEGEncoderFactory : DeviceFactory {
    MPEGEncoder new();
};

interface MPEGDecoderFactory : DeviceFactory {
    MPEGDecoder new();
};

interface FileSourceFactory : DeviceFactory {
    FileSource new();
};

interface FileSinkFactory : DeviceFactory {
    FileSink new();
};

interface ActiveNodeFactory : DeviceFactory {
    ActiveNode new();
};

interface ATMTransmitterFEPFactory : DeviceFactory {
    ATMTransmitterFEP new();
};

interface ATMReceiverFEPFactory : DeviceFactory {
    ATMReceiverFEP new();
};

interface TCPReceiverFEPFactory : DeviceFactory {
    TCPReceiverFEP new();
};

interface TCPTransmitterFEPFactory : DeviceFactory {
    TCPTransmitterFEP new();
};

interface NetManagerFactory : DeviceFactory {
    NetManager new();
};
```

```
interface MessageReceiver {  
    void send(in string msg);  
};
```

11.8 Lab Configuration

The following configurations were used in the lab:

zoppo:

- Intel PII 450MHz, 128MB,UW SCSI
- Intel Ether Express Pro Ethernet NIC
- Fore Runner LE ATM NIC
- Hauppauge WinTV Theater
- Matrox Millenium2+ RainbowRunner
- Optibase Videoplex
- NT Workstation 4.0

encoder:

- PII 300, 64MB, USCSI
- Fore Runnel LE ATM NIC
- Matrox Millenium2+ RainbowRunner
- Optibase Videoplex
- Optibase Forge MPEG-2 encoder
- NT Workstation 4.0

zwiebel:

- PIII 550, 256MB
- 3Com FastEtherlink XL Ethernet NIC
- Fore Runnel LE ATM NIC
- Elsa Gloria II graphic card, 64 MB texture memory
- Adaptec AHA2940U2W SCSI Adapter
- MPEG2 decoder card (RealMagic Hollywood+)
- Yamaha 4x2x24x CD burner
- Pinnacle DV300

- NT Workstation 4.0

smint:

- PII 450, 128 MB
- ATI Rage Pro
- UW SCSI

puma:

- PII 450, 128MB
- ATI Rage Pro
- UW SCSI
- NT Workstation 4.0

kallisto:

- PII 450, 256MB
- ATI Rage Pro
- UW SCSI
- RealServer 7.0
- Lotus Learning Space 4.0 Beta
- Saba Learning Online
- NT Server Enterprise 4.0 SP6a

ganymed:

- SGI O200 Server
- MIPS R10000, 180 MHz, 320 MB main memory
- Fore PCA 200E ATM NIC
- Fast Ethernet
- UW SCSI
- Mediabase 4.0
- Oracle 8.0.3
- IRIX 6.4

Fore ASX 200 WG ATM Switch

Fore ES 3810 Ethernet Switch

11.9 Abbreviations and Acronyms

API	Application Programming Interface
Arequipa	Application REQuested IP over ATM
ATM	Asynchronous Transfer Mode
A/V	audio/video
BOA	Basic Object Adaptor
BSD	Berkeley System Distribution
COPS	Common Open Policy Service
CORBA	Common Object Request Broker Architecture
DEN	Directory Enabled Network
DPE	Distributed Processing Environment
FEP	Flow Endpoint
IETF	Internet Engineering Task Force
ISO	International Standardization Organization
ISP	Internet Service Provider
JMF	Java Media Framework
OMG	Object Management Group
ORB	Object Request Broker
OSI	Open Systems Interconnection
QoS	Quality of Service
QoS-A	Quality-of-Service Architecture
RED	Random Early Discard
RSVP	Resource Reservation Protocol
RTP	Real-Time Transport Protocol
RTSP	Real-Time Streaming Protocol
SIP	Session Initiation Protocol
SNA	Systems Network Architecture
SPI	Service Provider Interface

TCP/IP	Transmission Control Program/Internet Protocol
TPI	Transport Provider Interface
UDP/IP	User Datagram Protocol/Internet Protocol
URL	Unified Resource Locator
WBT	Web-based Training

Bibliography

- [Alles95] A. Alles, "ATM Internetworking," Cisco Systems Inc. May 1995.
- [AlSaWr] W. Almesberger, S. Sasyan, and S. Wright, "Quality of Service in Communication APIs," *Proc. of IFIP International Workshop on Quality of Service (IWQoS'97)*, New York, 1997.
- [ATMF95] ATM Forum, "ATM User-Network Interface (UNI) Specification, Version 3.1," Prentice-Hall 1995.
- [ATMF96a] ATM Forum, "ATM Traffic Management Specification Version 4.0," ATM Forum af-tm-0056.000, April 1996.
- [ATMF96b] ATM Forum, "ATM User-Network Interface (UNI) Signalling Specification Version 4.0," ATM Forum af-sig-0061.000, July 1996.
- [ATMF96c] ATM Forum SAA Working Group, "Mapping of the ATM Forum SAA/API Semantic Description to the WinSock 2 API," ATM Forum ATM_Forum/96-0191R1, 1996.
- [ATMF96d] ATM Forum SAA Working Group, "Native ATM Services: Semantic Description Version 1.0," ATM Forum af-saa-0048.000, February 1996.
- [ATMF96e] ATM Forum SAA Working Group, "Use of XTI to Access ATM, Use of Sockets to Access ATM, ATM Transport Headers," ATM Forum ATM_Forum/96-1169, 1996.
- [ATMF96f] ATM Forum SAA Working Group, "WinSock 2 ATM Annex," ATM Forum ATM_Forum/96-0190R1, 1996.
- [ATMF97a] ATM Forum SAA/API Working Group, J. Harford, "Java ATM API from 50,000 Feet," ATM Forum ATM_Forum/97-1110, December 1997.
- [ATMF97b] ATM Forum SAA/API Working Group, T. Jespen, S. A. Wright, and Z. Zhang, "A Java Syntax implementation of the Native ATM Services," ATM Forum SAA/API Working Group ATM_Forum/97-0774, September 1997.
- [ATMF97c] ATM Forum SAA/API Working Group, T. Jespen and S. A. Wright, "A Java Syntax Instantiation of the Native ATM Services," ATM Forum ATM_Forum/97-0772, September 1997.
- [ATMF97d] ATM Forum SAA Working Group, T. Jespen and S. A. Wright, "A Java Syntax Mapping to the Native ATM Services," ATM Forum ATM_Forum/97-0773, September 1997.
- [ATMF97e] ATM Forum SAA Working Group, "Native ATM Services Data Link Provider Specific Addendum," ATM Forum ATM_Forum btd-saa-api-dlpi-01.00, 1997.

-
- [ATMF98] ATM Forum SAA/API Working Group, T. Jespen and J. Shaffer, "Java ATM API Description - Proposed Outline, Revision 1," ATM Forum ATM Forum/97-1044R1, February 1998.
 - [AuCaHa98] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A Survey of QoS Architectures," *Multimedia Systems Journal*, vol. May, 1998.
 - [BaPiRi98] M. Baldi, G. Picco, and F. Risso, "Designing a Videoconference System for Active Networks," *Proc. of Second International Workshop on Mobile Agents*, Stuttgart, 1998.
 - [BaHeRo95] I. Barth, T. Helbig, and K. Rothermel, "Implementierung multimedialer Systemdienste in CINEMA," *Proc. of Kommunikation in Verteilten Systemen (KiVS '95)*, Chemnitz, 1995.
 - [BeGe99] C. Becker and K. Geihs, "Generic QoS Specifications for CORBA," *Proc. of Kommunikation in Verteilten Systemen (KiVS '99)*, TU Darmstadt, 1999.
 - [BHL+99] J. Biswas, J.-F. Huard, A. Lazar, K. Lim, S. Mahjoub, L.-F. Pau, M. Suzuki, S. Torstensson, W. Weiguo, and S. Weinstein, "Application Programming Interfaces for Networks (Draft White Paper)," Working Group for IEEE P1520 01/28/99 1999.
 - [Booch94] G. Booch, *Object-oriented analysis and design with applications*, 2nd ed. Redwood City: Benjamin/Cummings, 1994.
 - [BoJaRu97] G. Booch, I. Jacobson, and J. Rumbaugh, *Unified Modeling Language Reference Manual*: Addison Wesley, 1997.
 - [BrClSh94] RFC 1633, R. Braden, D. Clark, and S. Shenker, "Integrated Services in the Internet Architecture: An Overview," June 1994.
 - [BrHo97] R. Braden and D. Hoffmann, "RSVP Application Programming Interface (RAPI)," draft-ietf-rsvp-rapi-00.ps, June 1997.
 - [BZBHJ97] RFC 2205, R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource Reservation Protocol (RSVP) - Version 1 Functional Specification," Internet Engineering Task Force (IETF), 1997.
 - [BETT97] A. Branchs, W. Effelsberg, C. Tschudin, and V. Turau, "Multicasting Multimedia Streams with Active Networks," International Computer Science Institute, Berkeley, CA, USA Technical Report TR-97-050, 1997.
 - [BrSt97] T. Braun and H. J. Stüttgen, "Implementation of an Internet Video Conferencing Application over ATM," *Proc. of IEEE ATM'97 Workshop*, Lisboa, Portugal, 1997.
 - [Brock95] K. Brockschmidt, *Inside OLE2*, Second ed: Microsoft Press, 1995.
 - [Camp96] A. T. Campbell, "A Quality of Service Architecture," in *Ph.D. thesis Computer Science*: Lancaster University, 1996.
 - [CCGH92] A. T. Campbell, G. Coulson, F. Garcia, and D. Hutchison, "A Continuous Media Transport and Orchestration Service," *Proc. of ACM SIGCOMM'92*, Baltimore, Maryland, USA, 1992.
 - [CHLL97] M. C. Chan, J.-F. Huard, A. A. Lazar, and K.-S. Lim, "On Realizing a Broadband Kernel for Multimedia Networks," , 1997.

-
- [ChSyLa97] B. S. S. Chatterjee, M. D. J. J. Sydir, and T. F. Lawrence, "Taxonomy for QoS Specifications," *Proc. of IEEE WORDS'97*, Newport Beach, CA, 1997.
 - [CHY+98] P. E. Chung, Y. Huang, S. Yajnik, D. Liang, J. C. Shih, C. Wang, Y. Wang, Bell Laboratories Lucent Technologies; Institute of Information Science, Academia Sinica, Taiwan; Microsoft Research, "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer," 1998.
 - [CBDW92] G. Coulson, G. Blair, N. Davies, and N. Williams, "Extensions to ANSA for Multimedia Computing," in *Computer Networks and ISDN Systems*, vol. 25, 1992, pp. 305 - 323.
 - [CoCaRo95] G. Coulson, A. T. Campbell, and P. Robin, "Design of a QoS Controlled ATM Based Communication System in Chorus," *IEEE Journal of Selected Areas in Communications (JSAC), Special Issue on ATM LANs: Implementation and Experiences with Emerging Technology*, 1995.
 - [Curtis97] D. Curtis, "Java, RMI and CORBA," OMG 1997.
 - [DAVIC95] DAVIC, "DAVIC 1.0 specification," Digital Audio-Visual Council December 1995.
 - [Delg96] L. Delgrossi, *Design of Reservation Protocols for Multimedia Communication*. Boston / Dordrecht / London: Kluwer Academic Publishers, 1996.
 - [DFFR99] L. Delgrossi, G. D. Fatta, D. Ferrari, and G. L. Re, "Interference and Communications among Active Network Applications," *Proc. of The First International Working Conference on Active Networks, IWAN'99*, Berlin, Germany, 1999.
 - [DHH+93] L. Delgrossi, C. Halstrinck, D. B. Hehmann, R. G. Herrtwich, J. Krone, C. Sandvoss, and C. Vogt, "Media Scaling for Audiovisual Communication with the Heidelberg Transport System," *Proc. of ACM Multimedia*, Anaheim, CA, USA, 1993.
 - [Ebe97] R. Eberhardt, "Experiences with a Transatlantic ATM Service," *Proc. of ATM Year '97 Europe*, London, 1997.
 - [Ebe98a] R. Eberhardt, "An Interactive Multimedia Distance Learning System using Distributed Object Computing Technology," *Proc. of IEEE Globecom 98*, Sydney, Australia, 1998.
 - [Ebe98b] R. Eberhardt, "Quality of Service (QoS) Support of Application Programming Interfaces (APIs)," *Proc. of 9th IEEE Workshop on Local and Metropolitan Area Networks*, Banff, Canada, 1998.
 - [EbRu99] R. Eberhardt and C. Rueß, "Eine Dienstgüteabbildungs- und -steuerungsarchitektur zur Gewährleistung unterschiedlicher Dienstgüteklassen für Ferntraining und -lernen," *Proc. of Kommunikation in Verteilten Systemen (KiVS '99)*, TU Darmstadt, 1999.
 - [ERM99] R. Eberhardt, C. Rueß, and J. Metzler, "Management and Control of Distributed Multimedia Devices and Streams Through Object-Oriented Middleware," *Proc. of 5th International Conference on IFIP Broadband Communications (BC '99)*, Hong Kong, China, 1999.

-
- [ERR98] R. Eberhardt, C. Rueß, and R. Rusnak, "Communication Application Programming Interfaces with Quality of Service Support," *Proc. of IEEE International Conference on ATM (ICATM '98)*, Colmar, France, 1998.
 - [ERS97a] R. Eberhardt, C. Rueß, and R. Sigle, "Multimediale Anwendungen in globalen ATM-Netzen," *Proc. of Kommunikation in Verteilten Systemen (KiVS '97)*, Braunschweig, 1997.
 - [ERSS97] R. Eberhardt, C. Rueß, C. Sinner, and H. Scherand, "Electronic Commerce - A Comparative Study of Web Based Database Access," *Proc. of International Switching Symposium (ISS'97)*, Toronto, 1997.
 - [EfSt98] W. Effelsberg and R. Steinmetz, *Video Compression Techniques*. Heidelberg: dpunkt-Verlag, 1998.
 - [FCIA00] Fibre Channel Industry Association, "Fibre Channel," <http://www.fibrechannel.com>, 2000.
 - [FeDe98] D. Ferrari and L. Delgrossi, "Charging for QoS," *Proc. of Sixth International Workshop on Quality of Service (IWQoS'88)*, Napa, CA, USA, 1998.
 - [FHBM97] S. Fischer, A. Hafid, G. v. Bochmann, and H. d. Meer, "Cooperative QoS Management for Multimedia Applications," *Proc. of 4th IEEE Int. Conference on Multimedia Computing and Systems (ICMCS'97)*, Ottawa, Canada, 1997.
 - [FIJa93] S. Floyd and V. Jacobson, "Random Early Detection Gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, pp. 397 - 413, 1993.
 - [Guan94] Guangxing, "A Model of Real-Time QoS for ANSA," APM Ltd., Cambridge, UK Technical Report APM.1151.00.04, 1994.
 - [GuFaVe98] E. Guarene, P. Fasano, and V. Vercellone, "IP and ATM integration Perspectives," *IEEE Communications Magazine*, pp. 74 - 80, 1998.
 - [HaScSc97] M. Handley, H. Schulzrinne, and E. Schooler, "SIP: session initiation protocol," Internet Engineering Task Force Work in progress, 1997.
 - [HaHoSt93] L. Hazard, F. Horn, and J. B. Stefani, "Towards the Integration of Real-Time and QoS Handling in ANSA," CNET CNET Report CNET.RC.ARCADÉ.01, June 1993.
 - [HHSSS91] D. B. Hehmann, R. G. Herrtwich, W. Schulz, T. Schütt, and R. Steinmetz, "Implementing HeiTS: Architecture and Implementation Strategy of the Heidelberg High Speed Transport System," *Proc. of Second International Workshop on Network and Operating System Support for Digital Audio and Video*, IBM ENC, Heidelberg, Germany, 1991.
 - [HILY96] J.-F. Huard, I. Inoue, A. A. Lazar, and H. Yamanaka, "Meeting QOS Guarantees by End-to-End QOS Monitoring and Adaption," *Proc. of Fifth International Symposium on High Performance Distributed Computing (HPDC-5)*, Syracuse, NY, 1996.
 - [IAL97] "Intel PC-RSVP via GQoS," Intel Architecture Labs Version 1.05 Developer's Guide, 1997.
 - [IEEE1394] 1394 Trade Association, "IEEE 1394 (Firewire)," <http://www.1394ta.org>, 2000.

-
- [IETF98] MMUSIC, "Multiparty Multimedia Session Control," <http://www.ietf.org/html.charters/mmusic-charter.html>, 1998.
 - [IT98] Iona Technologies, "OrbixMX - A Distributed Object Framework for Telecommunication Service Development and Deployment," White Paper, April 1998.
 - [ISO84] International Standardization Organization, "International Standard 7498 Information Processing Systems - Open Systems Interconnection -Basic Reference Model," ISO 7498, 1984.
 - [ISO96] ISO/IEC JTC1/SC29/WG11, "Information Technology - Generic Coding of Moving Pictures and Associated Audio: Digital Storage Media - Command and Control, ISO/IEC 13818-6 International Standard," ISO/IEC 1996.
 - [ISO98] International Standardization Organization, "Object Management Group Interface Definition Language (OMG IDL)," ISO/IEC 14750 International Standard, 1998.
 - [ITG98] ITG, "Begriffe der Nachrichtenverkehrstheorie," ITG 5.2-03, 1998.
 - [ITU91] International Telecommunication Union, "General Aspects of Quality of Service and Network Performance in Digital Networks, Including ISDN," ITU ITU-T I.350, 1991.
 - [ITU95] International Telecommunication Union, "Information Technology - Open Distributed Processing - Reference Model - Foundations," ITU-T and ISO/IEC ITU-T Recommendation X.902 | ISO/IEC 10746-2, 1995.
 - [ITU97a] International Telecommunication Union, "Information Technology - Open Distributed Processing - Reference Model - Quality of Service (QoS)," ITU-T and ISO/IEC ITU-T Recommendation X.905 | ISO/IEC 10746-5, 1997.
 - [ITU97b] International Telecommunication Union, "Information Technology - Quality of Service: Framework," ITU-T and ISO/IEC ITU-T Recommendation X.641 | ISO/IEC 13236, 1997.
 - [JaChJoÖv92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering, A Use Case Driven Approach*. Workingham: Addison-Wesley, 1992.
 - [Käp97] T. Käppner, *Entwicklung verteilter Multimedia-Applikationen*: Vieweg, 1997.
 - [KSW99a] M. Karsten, J. Schmitt, L. C. Wolf, and R. Steinmetz, "Cost and Price Calculation for Internet Integrated Services," *Proc. of Kommunikation in Verteilten Systemen (KiVS'99)*, Darmstadt, Germany, 1999.
 - [KSW99b] M. Karsten, J. Schmitt, L. C. Wolf, and R. Steinmetz, "Provider-Oriented Linear Price Calculation for Integrated Services," *Proc. of Seventh International Workshop on Quality of Service (IWQoS'99)*, London, England, 1999.
 - [KiMu96] B. Kinane and D. Muldowney, "Distributing Broadband Multimedia Systems Using CORBA," *Computer Communications*, vol. 19, pp. 13 - 21, 1996.
 - [KoNa97] K. Kowook and K. Nahrstedt, "QoS Translation and Admission Control for MPEG Video," 1997.

-
- [KKR97] H. Kröner, P. J. Kühn, and T. Renger, "Management von ATM-Netzen," *it+ti*, vol. 39, pp. 5 -14, 1997.
 - [KuWe95] A. J. Kunzman and A. T. Wetzel, "1394 High Performance Serial Bus: The Digital Bus for ATV," *IEEE Transactions on Consumer Electronics*, vol. 14, pp. 893-900, 1995.
 - [Lazar92] A. A. Lazar, "A Real-Time Control, Management and Information Transport Architecture for Broadband Networks," *Proc. of International Zurich Seminar on Digital Communications*, Vienna, Austria, 1992.
 - [LaBhLi95] A. A. Lazar, S. Bhonsle, and K. S. Lim, "A Binding Architecture for Multimedia Networks," *Journal of Parallel and Distributed Computing*, vol. 30, pp. 204-116, 1995.
 - [LTSC00] Learning Technology Standardization Committee, "Draft Standard for Learning Object Metadata LTSC 2000.
 - [MaBu99] P. Manhart and J. Bumiller, "Collaborative Web-based Learning in Organizations," *Proc. of WebNet'99*, Honolulu, Hawaii, 1999.
 - [MGRO97] D. McGrath, T. Rutt, and J. Ottensmeyer, "Control and Management of Audio/Video Streams," Object Management Group telecom/97-05-07, May 1997.
 - [MeEbe99] J. Metzler and R. Eberhardt, "Video Distribution over Intranets in a Telelearning Scenario," *Proc. of ICC'99*, Tokyo, Japan, 1999.
 - [MuSuSc99] S. Mungee, N. Surendran, and D. C. Schmidt, "The Design and Performance of a CORBA Audio/Video Streaming Service," *Proc. of HICSS-32 International Conference on System Sciences, minitrack on Multimedia DBMS and the WWW*, Hawaii, 1999.
 - [Na95] K. Nahrstedt, "An Architecture for End-To-End Quality of Service Provision and its Experimental Validation," in *Computer and Information Sciences*: University of Pennsylvania, 1995.
 - [NaSm95] K. Nahrstedt and J. Smith, "The QoS Broker," *IEEE Multimedia*, vol. March, 1995.
 - [Obra98] K. Obraczka, "Multicast Transport Protocols: A Survey and Taxonomy," *IEEE Communications Magazine*, pp. 94 - 102, 1998.
 - [Oest98] B. Oestereich, *Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modeling Language*. München: R. Oldenbourg, 1998.
 - [OMG95] "The Common Object Request Broker Architecture and Specification, 2.0 ed.," Object Management Group July 1995.
 - [OMG97] C. Sluman, J. Tucker, J. P. LeBlanc, and B. Wood, "Quality of Service (QoS) OMG Green Paper," Object Management Group ormsc/97-06-04.doc, June 6 1997.
 - [OMG00] Object Management Group, "CORBA," www.omg.org, 2000.
 - [OG97a] Open Group, "Data Link Provider Interface (DLPI), CAE Specification," The Open Group C614 ISBN 1-85912-196-9, February 1997.

-
- [OG97b] The Open Group, "Networking Services (XNS) Issue 5," The Open Group X/Open Document Number: C523, 1997.
 - [OrHa97] R. Orfali and D. Harkey, *Client/Server Programming with Java and CORBA*. New York: John Wiley & Sons, 1997.
 - [OMRW97] M. Ott, G. Michelitsch, D. Reininger, and G. Welling, "An Architecture for Adaptive QoS and its Application to Multimedia Systems Design," *Computer Communications*, 1997.
 - [Par92] RFC 1363, C. Partridge, "A Proposed Flow Specification," BBN, July 1992.
 - [RoBaHe94] K. Rothermel, I. Barth, and T. Helbig, "CINEMA - An Architecture for Distributed Multimedia Applications," in *Architecture and Protocols for High-Speed Networks*, O. Spaniol, A. Danthine, and W. Effelsberg, Eds., 1994, pp. 253-271.
 - [RuBlPrEdLo91] J. Rumbaugh, M. Blaha, W. Premerlan, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design*. Englewood Cliffs: Prentice-Hall, 1991.
 - [RuWoEbHo01] C. Rueß, M. Wolf, R. Eberhardt and J. Hördt, "Architektur für eine integrierte Training-Online-Plattform" to appear in *Proc. of Kommunikation in Verteilten Systemen (KiVS '01)*, TH Hamburg, 2001.
 - [SNIA] Storage Networking Association, "Storage Area Networks (SAN)," <http://www.snia.org>, 2000.
 - [SchZi95] C. Schmidt and M. Zitterbart, "Reservierung von Netzwerk-Ressourcen - Ein Überblick über Protokolle und Mechanismen," *PIK - Praxis der Informationsverarbeitung und Kommunikation*, vol. 18, pp. 140 - 147, 1995.
 - [Sch93] D. C. Schmidt, "The ADAPTIVE Communication Environment - An Object-Oriented Network Programming Toolkit for Developing Communication Software," *Proc. of 12th Sun User Group (SUG) conference*, San Jose, CA, USA, 1993.
 - [SBLMP97] D. C. Schmidt, R. Bector, D. L. Levine, S. Mungee, and G. Parulkar, "An ORB Endsistem Architecture for Statically Scheduled Real-Time Applications," *Proc. of IEEE Workshop on Middleware for Real-Time Systems and Services*, San Francisco, CA, 1997.
 - [SGHP97] D. C. Schmidt, A. S. Gokhale, T. H. Harrison, and G. Parulkar, "A High-Performance End System Architecture for Real-Time CORBA," *IEEE Communications Magazine*, pp. 72 - 77, 1997.
 - [ScLeMu97] D. C. Schmidt, D. L. Levine, and S. Mungee, "The Design of the TAO Real-Time Object Request Broker," *Computer Communications*, 1997.
 - [SWKS99] J. Schmitt, L. C. Wolf, M. Karsten, and R. Steinmetz, "A Taxonomy of Interaction Models for Internet and ATM Quality of Service Architectures," *Telecommunication Systems Journal*, 1999.
 - [Sch97] H. Schulzrinne, "A comprehensive multimedia control architecture for the Internet," *Proc. of 7th Int. Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'97)*, St. Louis, Missouri, USA, 1997.
 - [SCFJ96] RFC 1889, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobsen, "RTP: A Transport Protocol for Real-Time Applications," IETF, January 1996 1996.

- [ScRaLa98] H. Schulzrinne, A. Rao, and R. Lanphier, "Real-Time Streaming Protocol (RTSP)," Internet Engineering Task Force (IETF) draft-ietf-mmusic-rtsp-08.ps, January 15 1998.
- [SLCL99] N. Semret, R. R.-F. Liao, A. T. Campbell, and A. A. Lazar, "Market Pricing of Differentiated Internet Services," *Proc. of Seventh International Workshop on Quality of Service (IWQoS'99)*, London, England, 1999.
- [SCEH96] S. Shenker, D. Clark, D. Estrin, and S. Herzog, "Pricing in Computer Networks: Reshaping the Research Agenda," *ACM Computer Communication Review*, 1996.
- [ShPaGu97] RFC 2212, S. Shenker, C. Partridge, and R. Guerin, "Specification of the Guaranteed Quality of Service," 1997.
- [Ste99] R. Steinmetz, *Multimedia-Technologie; Grundlagen, Komponenten und Systeme*. Berlin, Heidelberg: Springer, 1999.
- [StHe91] R. Steinmetz and R. G. Herrtwich, "Integrierte verteilte Multimedia-Systeme," *Informatik Spektrum*, vol. 14, pp. 280-282, 1991.
- [StNa95] R. Steinmetz and K. Nahrstedt, *Multimedia: Computing Communications and Applications*: Prentice-Hall, 1995.
- [StRüRa90] R. Steinmetz, J. Rückert, and W. Racke, "Aktuelles Schlagwort: Multimedia-Systeme," *Informatik Spektrum*, vol. 13, pp. 280-282, 1990.
- [StWo97] R. Steinmetz and L. C. Wolf, "Quality of Service: Where are We?," *Proc. of IFIP Fifth International Workshop on Quality of Service (IWQOS'97)*, Columbia University, New York, USA, 1997.
- [Sun99] Sun, "Java Media Framework 2.0 API Guide, Version 0.8," September 3, 1999 1999.
- [Sun97] Sun, SGI, and Intel, "Java Media Players Specification Version 1.0.3," November 6 1997.
- [SuSo97] SunSoft, "Solstice Bandwidth Reservation Protocol 1.0 Programming Guide," SunSoft March 1997.
- [Tas97] J. Tassel, "Quality of Service (QoS) adaptation using Reflective Java," in *Computing Laboratory*: University of Kent at Canterbury, 1997.
- [TaBrSm97] J. Tassel, B. Briscoe, and A. Smith, "An End to End Price-based QoS Control Component Using Reflective Java," *Proc. of 4th COST Workshop*, Lisbon, 1997.
- [TINA95] "The QoS Framework," TINA-C Internal Technical Report, 1995.
- [Top90] RFC 1190, C. Topolcic, "Experimental Internet Stream Protocol, Version 2 (ST-II)," October 1990.
- [USB] USB Implementers Forum, "Universal Serial Bus (USB)," <http://www.usb.org>, 2000.
- [Vin97] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogenous Environments," *IEEE Communications Magazine*, 1997.

-
- [VWHW98] C. Vogt, L. Wolf, R. G. Herrtwich, and H. Wittig, "HeiRAT - Quality of Service Management for Distributed Multimedia Systems," *Multimedia Systems Journal*, vol. 6, 1998.
- [WSRR98] S. Weinstein, M. Suzuki, J. P. Redlich, and S. Rao, "A Distributed Object Architecture for QoS-Sensitive Networking," NEC, Technical Report 98-R-003, 1998.
- [WSG96] Winsock Group, "Windows Sockets Version 2, Protocol Specific Annex, Version 2.0.3," <<http://www.sockets.com/winsock2.htm>>, 1996.
- [WSG97a] Winsock Group, "Windows Sockets Version 2, Application Programming Interface (API), Version 2.2.1" <<http://www.sockets.com/winsock2.htm>>, May 1997.
- [WSG97b] Winsock Group, "Windows Sockets Version 2, Service Provider Interface (SPI), Version 2.2.1," <<http://www.sockets.com/winsock2.htm>>, May 1997.
- [Wolf96] L. Wolf, *Resource Management for Distributed Multimedia Systems*. Boston / Dordrecht / London: Kluwer Academic Publishers, 1996.
- [Wolf98] L. C. Wolf, "Resource Management in Multimedia Systems," in *Handbook of Multimedia Computing*, B. Furht, Ed. Boca Raton, FL, USA: CRC Press, 1998.
- [WoGrSt97] L. C. Wolf, C. Griwodz, and R. Steinmetz, "Multimedia Communication," in *Proceedings of the IEEE*, vol. 85, 1997, pp. 1915-1933.
- [Wro97b] RFC 2211, J. Wroclawski, "Specification of the Controlled-Load Network Service," Internet Engineering Task Force, September 1997.